SHARPパソコンテレビ

X1シリーズ(X1・X1C・X1D)

X1テクニカルマスター

ストラットフォードC.C.C. 著

日本ソフトバンク

SHARPパソコンテレビ

X1シリーズ(X1・X1C・X1D)

X1テクニカルマスター

ストラットフォードC.C.C. 著

日本ソフトバンク

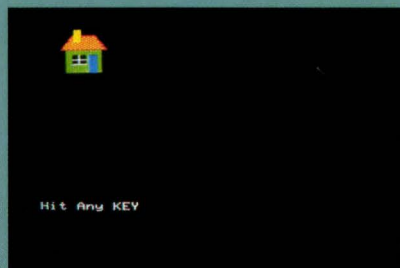
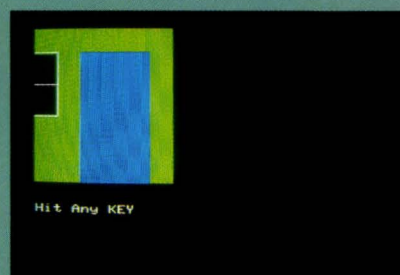
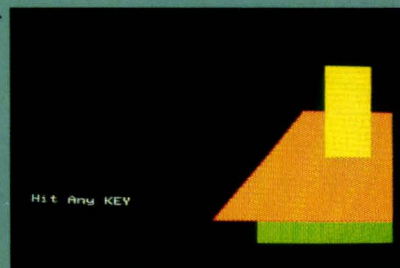
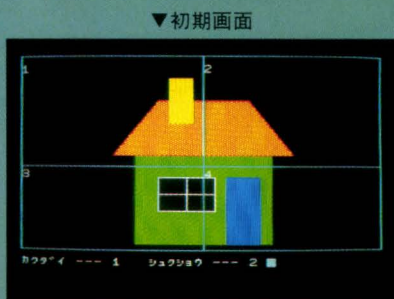
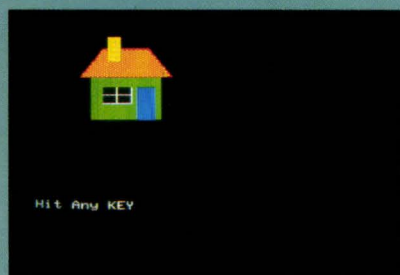
X1 カラーグラフィック画面

口絵カラーページに掲載しましたカラーグラフィック画面のプログラムは本書巻末「付録」にあります。
本文をよく読んだ後にプログラムの意味を考え、復習をしながら実際に打ち込んで見てください。

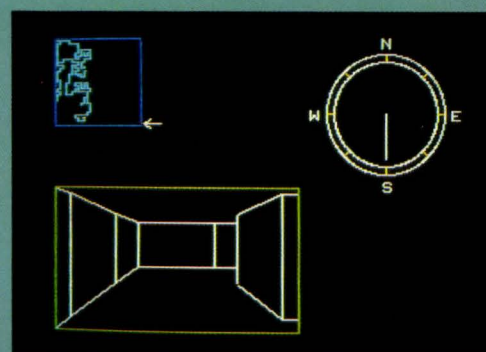
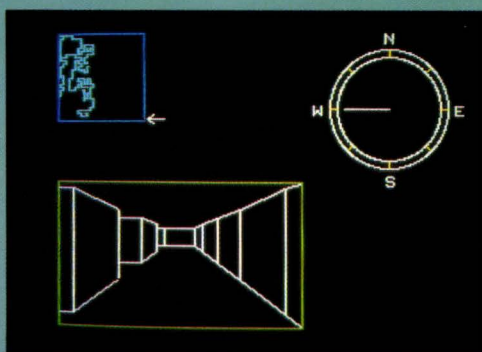
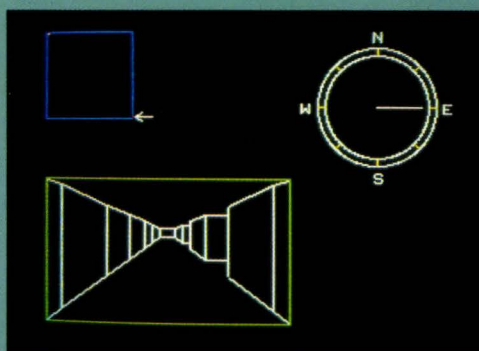
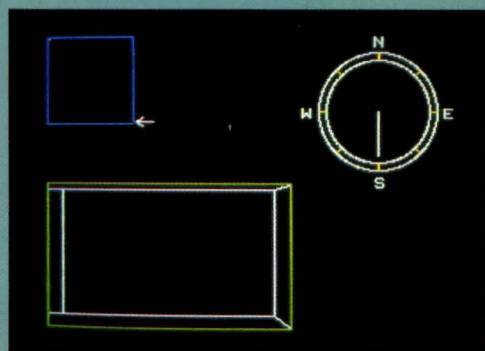
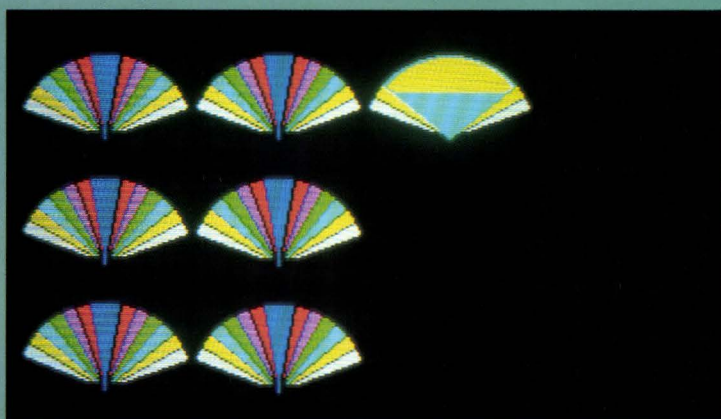
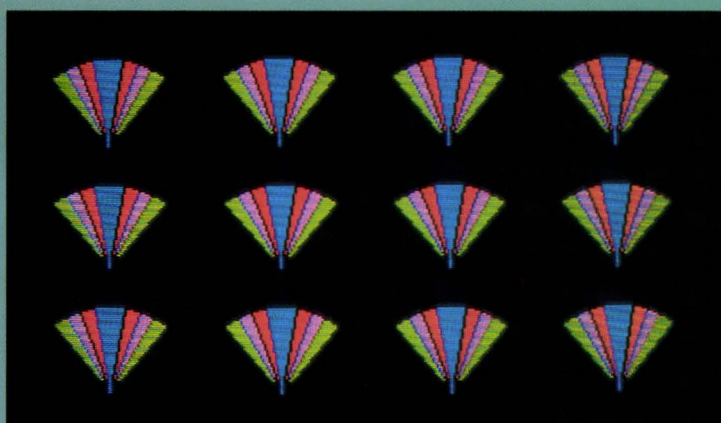
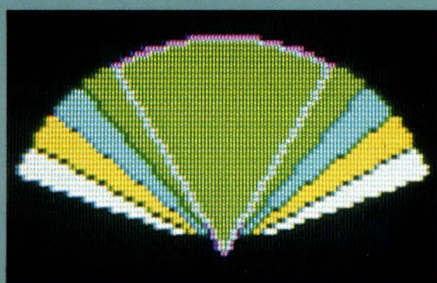
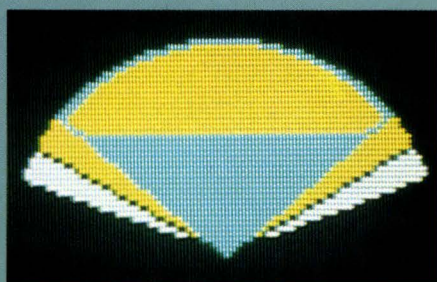
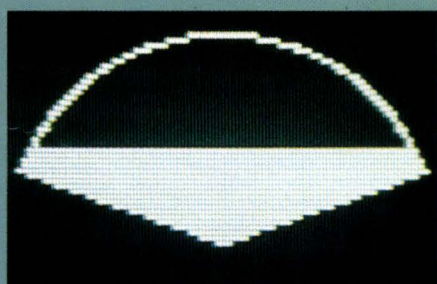
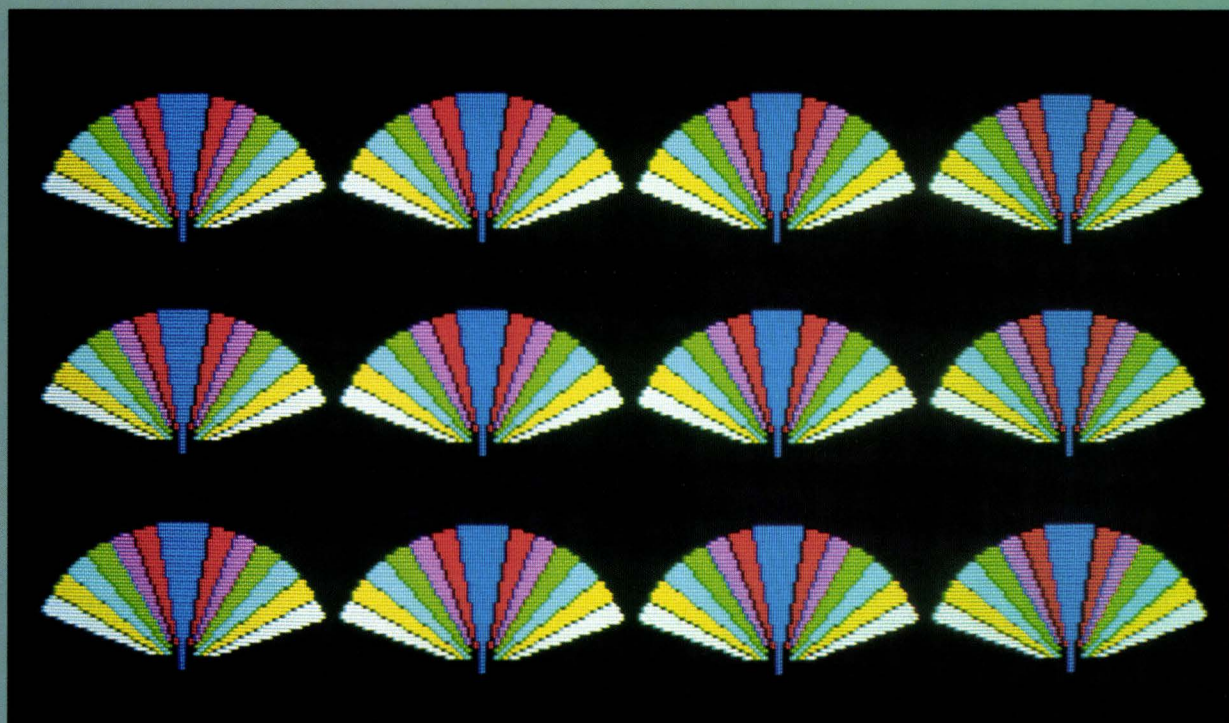
HOUSE

初期画面の家を、拡大、縮小させることができます。

◀縮小

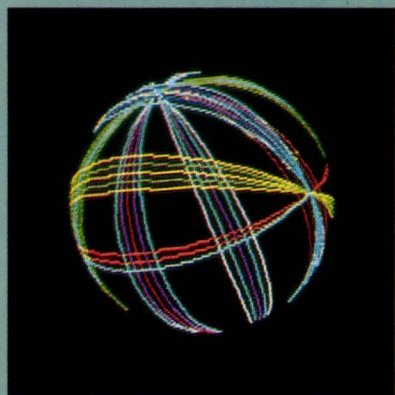
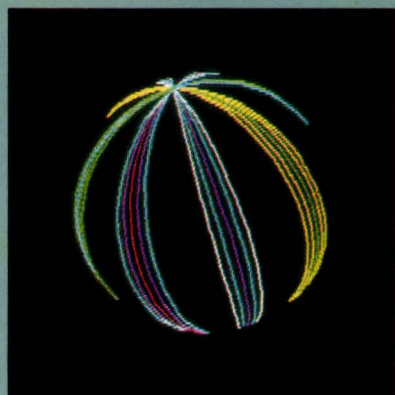
▼初期画面

拡大▶

MAZE

3次元立体迷路です。プログラムを打ち込んだら、
今度は、何分で迷路を抜け出せるか挑戦してください。

12個のカラフルな扇が、１つずつ画面にあらわれ、開いたり閉じたり、
BGMもついたプログラムです。

百人一首

百人一首と銘打ったプログラムです。
十二ひと衣の美しい平安美人があらわれます。

手まり

ちょっと時間はかかりますが、きれいな手まりができます。
本文をよく読んで勉強すればもう少し細かい模様が描けます。

は　し　が　き

シャープから「パソコンテレビＸ１」が発表されたのは，82年の暮れでした。先進的なグラフィック機能，自由に定義できるキャラクタ・ジェネレータ，優れたサウンド機能など，パソコンとして考えられるほとんどの機能が盛り込まれたうえ、世界でも初のテレビ放送とコンピュータ画像とのスーパーインポーズ機能をたずさえての登場です。パソコンとテレビが一体となった「パソコンテレビＸ１」が衝撃的なデビューを飾ったのでした。

Ｘ１は、単にパソコンとテレビを組み合わせただけの複合商品ではなく，ビジュアルな面での総合機器としての性格と特徴を持っています。パソコンのキーボードからテレビの音声やチャンネルを操作できる。パソコンのタイマーでテレビをコントロールできる。テレビ放送の画面にパソコンからの文字を重ねて表示できる。デジタルテロッパと組み合わせてビデオの編集ができる，などなどです。まさにホームコンピュータ時代の幕あけを告げる，斬新なハードウェアの誕生をＸ１に見たわけです。Ｘ１はパソコンの革命児だというのが第一印象でした。

Ｘ１のハードウェアの構成には、高度で斬新な技術的発想がたくさん盛り込まれています。そして，そのハードウェアの多彩な機能をあますところなく発揮させるために開発されたのが「SHARP Hu-BASIC」です。

本書では，この「SHARP Hu-BASIC」をわかりやすく解説していきます。初めてコンピュータにふれる方から，ある程度BASICを知っていて，Ｘ１独自の機能を活用しようとしている方までを対象にしました。

「習うより慣れろ」ということばをよく耳にしますが，このことばはまさにパソコンをマスターするためにあるようなものです。最初はBASICの命令の意味などわからなくても、実際にＸ１を動かして間違いを繰り返しながら親しんでいくうちに，使い方も意味も自然に身につきます。特に，初めてパソコンにふれるという方は，説明文だけで理解しようとせずに，何か疑問に思ったら，すぐにキーボードから打ち込んで試して見るようおすすめします。本書では，少しづつ確実に知識と技術が身につくように，実際の例をたくさんあげながら説明を進めていくようにしました。

Ｘ１は非常に多くの機能を持っているので，それらをコントロールするためのBASICの命令も多岐にわたっています。ある程度BASICをマスターしている方は，グラフィックやサウンドなど，興味を持てるところからページを開いてください。楽しみながらＸ１の機能が理解できます。

なお，記述内容については，誤りのないよう十分配慮したつもりですが，お気づきの点や，ご意見などがございましたら，ご教示くださるようお願いいたします。

また，本書の出版にあたって，ひとかたならぬご指導，ご協力をいただいたシャープ株式会社電子機器事業部，ならびに株式会社日本ソフトバンク出版部の皆様方，ストラットフォードの福井，伊東，尾林氏には，ここに誌面をお借りして心からのお礼を申しあげる次第です。

1983年８月

著者代表　須川　晴男

目　次

付　録　263

1

X1にさわってみないか？

1-1　パソコン、マイコンてなんだ?

●パソコン，マイコンとは？

近頃，「ＯＡ」とか「ＦＡ」などという言葉が新聞紙上をはじめ世間をにぎわしています。その中心になっているのが「パソコン」――パーソナルコンピュータです。

また，「マイコンブーム」などという言葉もよく聞かれます。家庭電気製品の中にも「マイコン内蔵」をキャッチフレーズにしたものがたくさん目につくようになりました。

さて，このパソコンとかマイコンというのは具体的にどんなものなのでしょうか。

「マイコン」と「パソコン」とはよく混同して使われています。たとえば「パソコン」を販売しているお店を「マイコンショップ」と言ったり，「パソコン」の使い方を教える学校も「マイコンスクール」と呼んだりします。この本で取り上げる「パソコンテレビＸ１」は，「マイコン内蔵エアコン」や「マイコン制御直火炊き」の電気釜とどう違うのでしょうか。

「そんなことはわかりきっている。エアコンは部屋の温度を調節するもので，それを上手にコントロールするために『マイコン』を利用しているのだ」と，誰もが思うでしょう。

ところで，「マイコン」というのは，手軽に使えることから，自分のコンピュータという意味での「my computer」の略だと思う人がいるかもしれませんが，本当は「マイクロ・コンピュータ（micro computer)」の略語なのです。このマイクロ・コンピュータは小さなチップ（小片）１枚で大きさは５～６ミリ四方，小指の先にのるくらいのサイズですが，一昔前の大型コンピュータと同じ機能を持っています。このチップのことを部品名でLSI(エルエスアイ，Large Scale Integrated circuit）と呼んでいます。日本語になおすと，「大規模集積回路」となりますが，言いにくいのでふつうはLSIと呼んでいます。このLSIに仕事の手順であるプログラムを書き込んだものを本来マイコンというのですが，最近では意味が広がり，システムとしてLSIを組み立てたものまでマイコンと呼ぶようになっています。

個人で買うことができるコンピュータは最初，「コンピュータ」の働きを学習したり，理解するためのものでしかありませんでした。プリント基板もむき出しになっていましたが，それでも立派にコンピュータと呼べるものでした。

このコンピュータが目ざましく発達して，現在見られるようなディスプレイやキーボードと一体になり，机の上に置いて使えるコンピュータとなったのです。このような形になったものを改めて「パソコン」――パーソナルコンピュータ（personal computer）と呼ぶようになりました。

●BASICでコンピュータと話そう

コンピュータというものを簡単に言い表わすと，あらかじめ決められた仕事を，手順どおりに忠実に繰り返し行なうものということになります。

朝起きてから夜寝るまでの一日の「手順」を書き表わしてみましょう。たとえば図１-１のよ

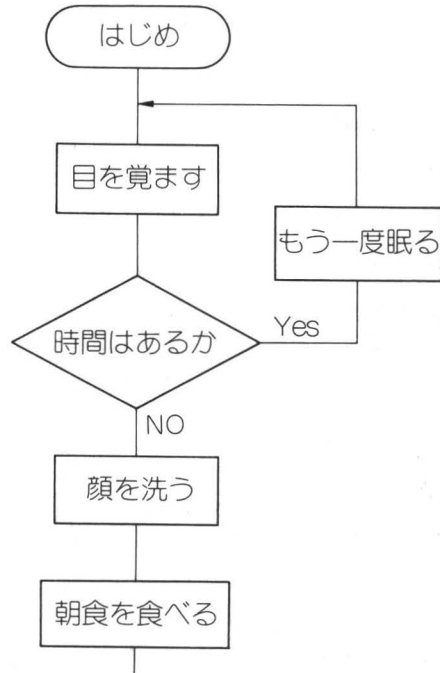
うになるでしょう。

コンピュータは，このような手順を忠実に守って仕事をします。人間のように寝ぼうをしたり，歯をみがきわすれることは決してありません。また，この図の中にあるような判断をすることもコンピュータはやってのけるのです（このような一連の手順を「プログラム」とよびます)。

マイコン内蔵のエアコンや電気釜では，それぞれ決められた仕事に対して一番具合のいい条件にするようプログラムされた「マイコン」が組み込まれており，これらのマイコンは，エアコンや電気釜それぞれの仕事専用に設計されています。ですから使う側が室内の温度を測ってエアコンをストップさせる操作をしたり，お米の炊き具合をチェックしたりする必要はないのです。

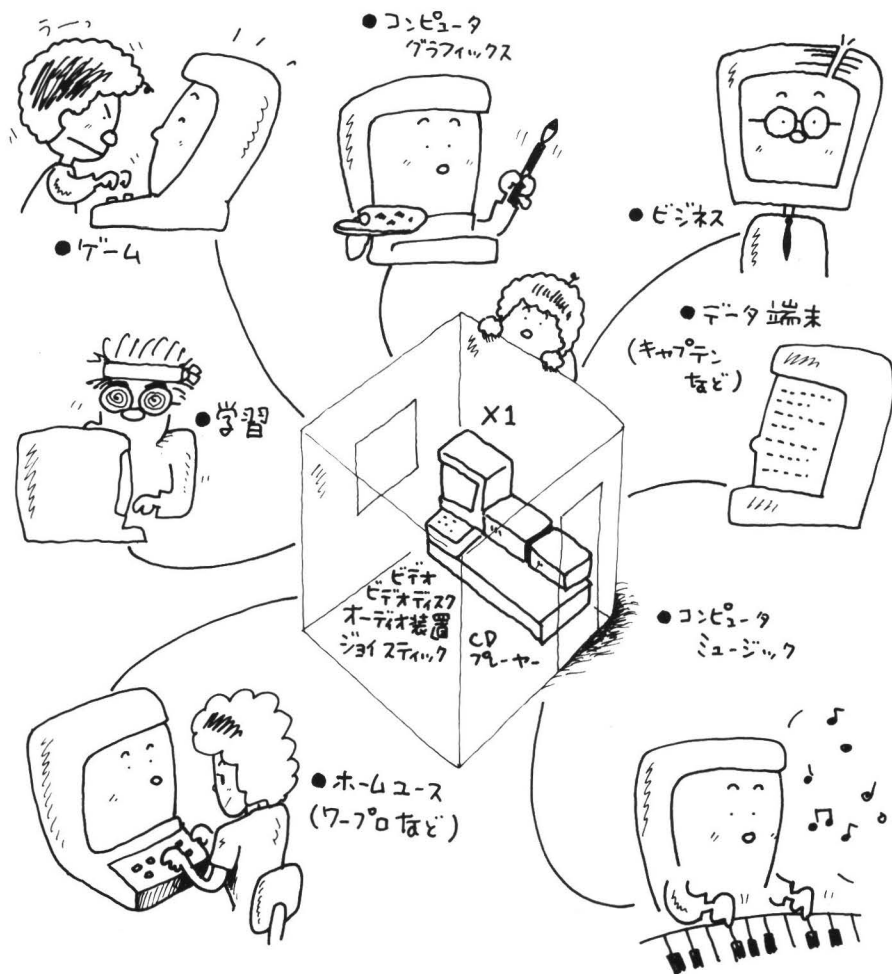
これに対して「パソコン」では，どんな仕事をさせるかを，使う人がコンピュータに覚え込ませる必要があるわけです。これは逆に言えば，使う人それぞれが，自分に都合のいいような仕事をさせることができるのです。パソコンに仕事の手順を指示して，電気釜を「マイコン内蔵」と同じ働きをするようにコントロールすることもできるわけです。

パソコンを動かすために必要な「プログラム」の作り方をマスターすれば，パソコンは心強いあなたのパートナーになります。パソコンテレビX 1 は，パソコンの中でも特に活動範囲の広いものです。人間がやると何時間もかかる仕事をわずかの時間で仕上げてくれますし，ゲームを楽しむこともできます。音楽も聞かせてくれます。また，英語や算数の勉強にも使えます。テレビ番組の時刻やチャンネルの予約をしたり，ビデオにつないで楽しんだりすることもできます。X 1 は，このように多くの機能で私たちの生活を彩ってくれます。

そのためには，X 1 とあなたがお互いの意思を通じ合わせることが必要になってきます。意思を通じ合わせるためには，コンピュータと話をするための新しい言葉を覚えなければなりません。こう書くと，英語の授業など思い出して，「むずかしそうだ」と思うかもしれません。でも，どうか安心してください。X 1 を自由自在に操るために私たちがこれから勉強する言葉は，ごく簡単な英語の単語の組み合わせだけでできています。この言葉を「BASIC」と呼びます。少しずつコンピュータを動かしているうちにBASICも，コンピュータの使い方も自然に身についてきますから，焦らず，あきらめず，少しずつゆっくりとマスターしていきましょう。

1-2　パソコンテレビX1で あなたの生活はこう変わる

パソコンとテレビが一緒になると，いろいろな可能性が未来に向かって広がります。

ゲームを楽しんだり，少し高級にシミュレータとしてX１を使ったり，ビデオの編集に使ったり，勉強にだって使えます。ビジネスにも立派に役立ち，ワードプロセッサやコンピュータグラフィックス，コンピュータミュージックの道具としてもX１は活躍します。データ通信の端末に使われるようになれば，楽しい未来の暮らしの中で，X１は本当に主役になるに違いありません。

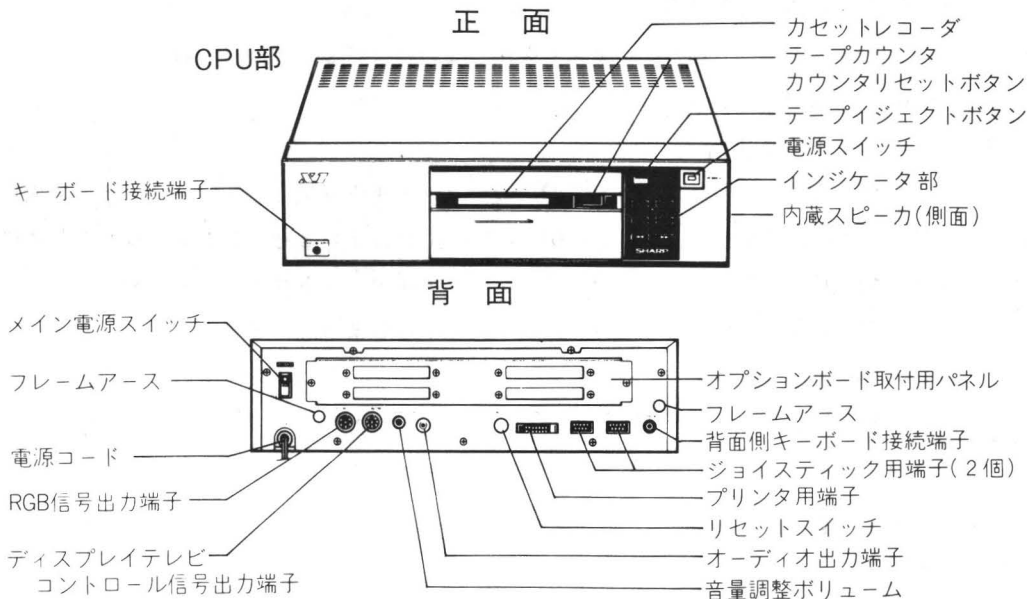
いえ，主役はX１を使いこなすみなさんです。X１はいろいろな場面でみなさんの暮らしのお手伝いをするだけです。X１でみなさんの暮らしはどこまで変わるか，どこまで変えられるかは，みなさんの力と希望しだいなのです。

1-3 X１の身体検査

●X１の身体検査

実際にX１の各部分を見ていくことにしましょう。

X１は，テレビの部分（CZ－800D）とコンピュータ部分（CZ－800C）の２つで構成されています。コンピュータの部分は，四角い箱のようなコンピュータ本体と，タイプライタに似たキーボードに分かれています。

中身も見えない四角い箱が，人間でいえば頭脳にあたります。これが目や耳や手足にあたるディスプレイとキーボードなどの中心になって，それらをコントロールします。この四角い箱が本体の部分で，「Central Processing Unit（中央演算処理装置）」の頭文字をとって，CPU部とも呼ばれます。

〔図１－２〕

CPU部の前と後ろは図１－２のようになっています。前から見るとちょうどカセットデッキのように見えます。ですが，音楽を聞くためのカセットデッキとはちょっと違います。ここには，コンピュータプログラムを録音したカセットテープを入れて使います。カセットのドアを開けるには，電源を入れて，本体の右側の「EJECT」ボタンを押します。EJECTボタンの下には，いろいろなインジケータランプがついており，その横の赤い枠の中が電源スイッチです。コンピュータ本体部の電源の「ON/OFF」にはこのスイッチを使います。

本体の後ろ側にも電源スイッチがあります。背面のこのスイッチがX１のメイン電源スイッチですが，普段は使わず，ONにしておきます。というのは，コンピュータの中にあるテレビのタイマー機能を保つためなのです。メイン電源スイッチを切ると，セットされていたタイマー

も消えてしまいます。

前面の左端には小さなジャックがありますが，ここにはキーボードからのびたケーブルのプラグをさし込みます。背面にも同じジャックがあります。X１の置き方や使う場所によって２つのジャックを使い分けられるようになっているのです。キーボード専用のジャックですから，イヤホンなどのプラグを差し込んではいけません。

本体の背面にはいろいろな端子（さし込み口，ジャック）があります。これらはテレビをはじめとして，ビデオやプリンタなどの機器との接続に使います。右端はキーボードを接続するための端子です。その横には，ジョイスティック用の端子が２つあります。ゲームを楽しむとき，この端子にジョイスティックをつなげば，ゲームセンターのテレビゲームとまったく同じようにX１で遊ぶことができます。

次はプリンタ用端子です。プリンタは，コンピュータからの指令によって文字を打ち出す機械です。その左の黒い押しボタンはリセットスイッチといいます。何かの理由でコンピュータの動作をコントロールできなくなってしまったときにだけ使います。

「AUDIO OUT」は，X１が持っているシンセサイザが作り出す音を取り出すための端子です。ステレオなどの「AUX」という端子と接続すると，ダイナミックなコンピュータ・ミュージックを楽しむことができます。ステレオなどと接続しない場合でも，本体に内蔵されたスピーカから音が出るようになっています。そのときの音量をコントロールするボリュームつまみが「AUDIO OUT」の左についています。

X１では，ディスプレイテレビ（CZ-800D）のチャンネルや音量などを，キーボードの操作でコントロールすることができます。X１の標準BASICにはテレビをコントロールするための命令もあります。ディスプレイテレビの音量やチャンネルをコントロールするための信号を送り出すのが，「TV CONTROL」端子です。付属のケーブルを使ってディスプレイテレビと接続します。その左の同じような端子は，ディスプレイテレビの画面にコンピュータの映像を映し出させる信号を送り出す端子です。「RGB」といって色の三原色の赤，緑，青の信号を別々に分離してここから出します。これも付属のケーブルでテレビと接続します。これらの２つの端子の形はそっくりですが，ピンの数は異なっています。あやまった接続ができないようになっているのです。

これらの端子にケーブルをさし込むときは，必ず本体の電源を切った状態で行なってください。

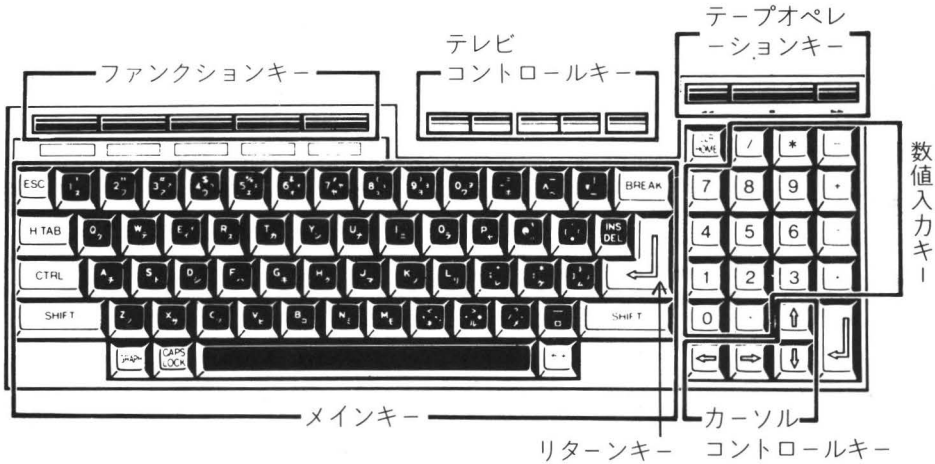
●キーボード

キーボード部分は，あなたとコンピュータとをつなぐ重要な部分です（図１-３）。

キーボードにはたくさんのキーが並んでいますが，それらを大きく５つのグループに分けることができます。

まずメインキー部を見てみましょう。アルファベットや記号のキーが並んでいます。BASICの命令などはここから打ち込んでいきます。アルファベットの文字がバラバラに並んでいるように見えますが，これは，タイプライタのキーの配列にならったもので使う頻度の多いキーを中央に集めて使いやすくしているのです。

メインキーの上にあるのはファンクションキーです。ひんぱんに使うBASICの命令語が，こ

〔図1-3〕

れらのキー1つでコンピュータに送り込めるようになっています。

右側の，数字などを集めた部分を数値入力キー，またはテンキーと呼びます。数字や計算式を打ち込むときに便利です。「＋」，「－」，「＊」，「／」などの計算記号や，カーソルを移動させるキーもここにあります。コンピュータでは，乗算の「×」の記号を「＊」（アスタリスク），除算の「÷」を「／」（スラッシュ）として使います。

テンキーの上は，カセットコントロールキーで，X 1本体のカセットデッキをこのキーで操作します。「再生」や「録音」のキーがない点が一般のカセットデッキのボタンとは異なります。これは，X 1では，録音や再生といった操作をBASICの命令で行なうからなのです。

その左の銀色のキーは，ディスプレイテレビのコントロールを行なうキーです。この5つのキーは，ディスプレイテレビについている「VOLUME」や「CHANNEL」などのスイッチとまったく同じ働きをします。ディスプレイテレビのスイッチをキーボードに引き出してきた形ですから，他のキーとは少し性格が違います。

1-4 X1と愉快な仲間たち

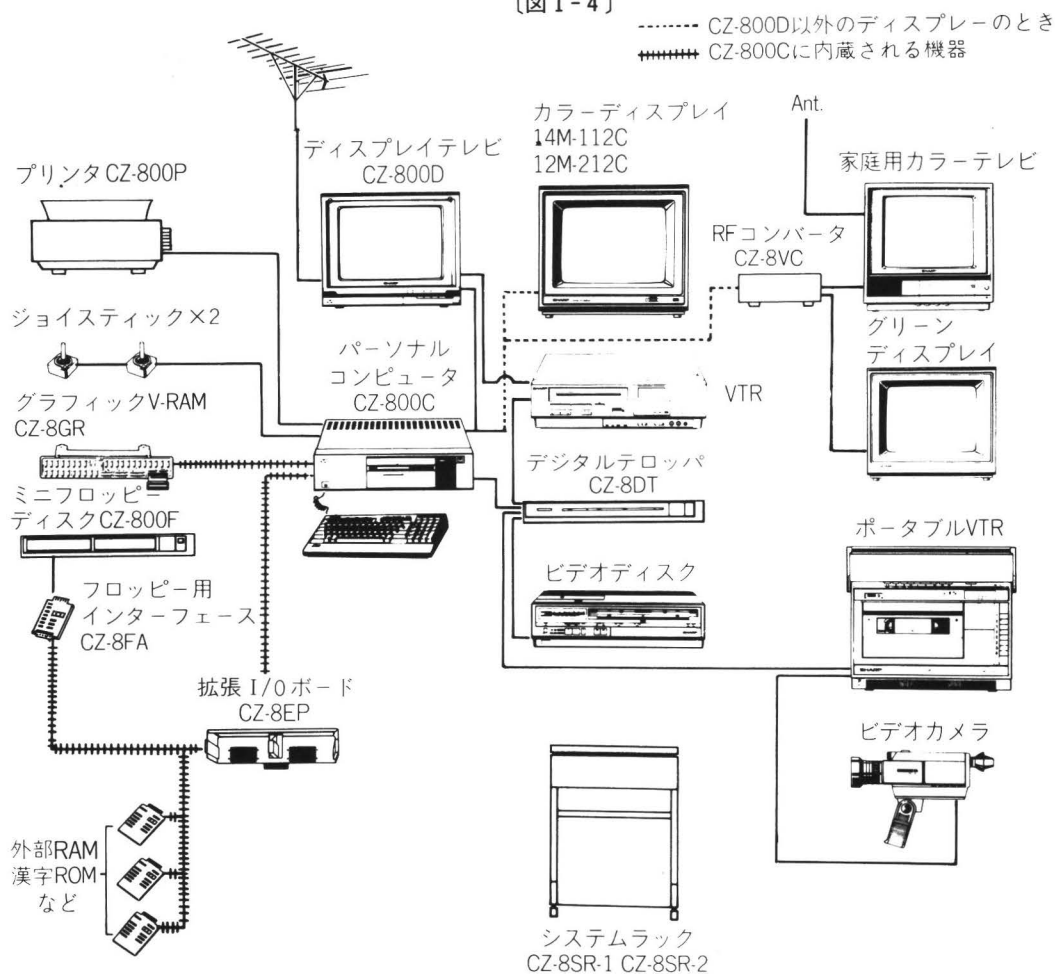
●X1の強い味方

X1は，本体とディスプレイテレビの組み合わせだけでも1台のコンピュータとして使うことができます。

皆さんがX1のBASICを十分にマスターし，さまざまに使いこなしていけるようになると，計算の結果を紙の上に残しておきたいとか，データやプログラムの出し入れをスピーディに行ないたいと思うようになるでしょう。

また，X1のディスプレイはビデオと接続することができますし，さらにコンピュータの機能を組み合わせることもできます。

こんなときに活躍するX1の仲間たち，周辺機器を紹介しましょう（図1-4）。

〔図1-4〕

1．プリンタ (CZ-800P)

X1にいろいろな計算処理をさせるときなど計算結果やメッセージがとても多くなり，ディスプレイテレビの画面に収まりきらなくなってしまいます。だからと言って，画面に表示された文字をいろいろ紙に書き写していたのでは能率も悪く，スマートではないし，だいいち誤りのもとになります。また，打ち込んでいったプログラムが長過ぎて，画面に収まらなくなったときでも，そのプログラムの全体を見なければならないことがしばしばあります。このような場合，プリンタがあると非常に便利です。

プログラムの内容（これをプログラムリストといいます）やデータを紙に打ち出すこともできます。X1の専用プリンタCZ-800Pは，X1で使われているすべての文字や記号を打ち出すことができます。また，グラフィックで描いた画面をそっくりコピーすることもできます。ゲームの名場面を残しておくこともできます。プリンタは専用のケーブルを使って本体後部のプリンタコネクタと接続します。

2．フロッピーディスクドライブ (CZ-800F)

X1本体にはカセットデッキが内蔵されています。プログラムやデータはカセットテープに記録したり，読み出したりできます。X1のカセットテープへの書き込みや読み出しの速度は，他のパソコンに比べ，非常に高速です。それでも，BASICテープの読み込みには2分40秒ほどかかってしまいます。

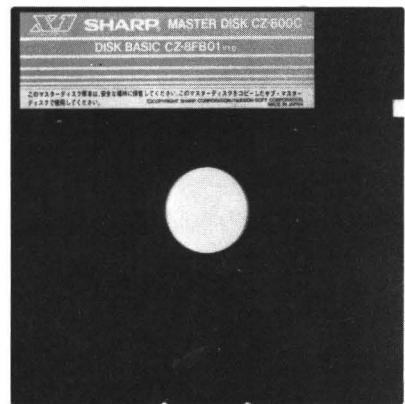
フロッピーディスクを使うと，こういった仕事のスピードが大変速くなります。それだけでなく，テープでは絶対にできないことが可能になります。

〔写真1-1〕

写真1-1のように，四角いケースの中に薄いレコードのようなものが入っているのがフロッピーディスクです。X1でカセットとフロッピーディスクを使うときの違いは，ちょうどテープとレコードの違いにあてはまります。たとえば，レコードの3曲目を聞きたいときには，その曲のはじまりの部分にレコードプレーヤのアームを持っていって針をおろせば良いですね。それに対して，テープではははじめの曲から順番に早送りしていって，目的の曲を見つけださなければなりません。

カセットテープの場合にはプログラムやデータははじめから終わりまで順番に記録されています。このような記録方法をシーケンシャルファイルと呼びます。フロッピーディスクを使うと，ちょうどレコードの3曲目に針をおろすように，希望のプログラムやデータからいきなり作業をはじめることができます（ランダムファイル）。

フロッピーディスクドライブを使用するにはX1の本体内部にオプションの**拡張I/Oポート**(CZ-8EP）を取りつけ，さらにインターフェースを設置する必要があります。

インターフェース（interface）とは，コンピュータ本体に新しい入出力装置をつなぐときなどに，2つの装置をつないで十分にその機能が働くようにするための，2つの装置間の共有部

分というものです。簡単にいえば２つのシステムを接続させるものということになります。Ｘ１には，**フロッピー用インターフェース**（CZ-８FA）がありますから，それをセッティングしてください。

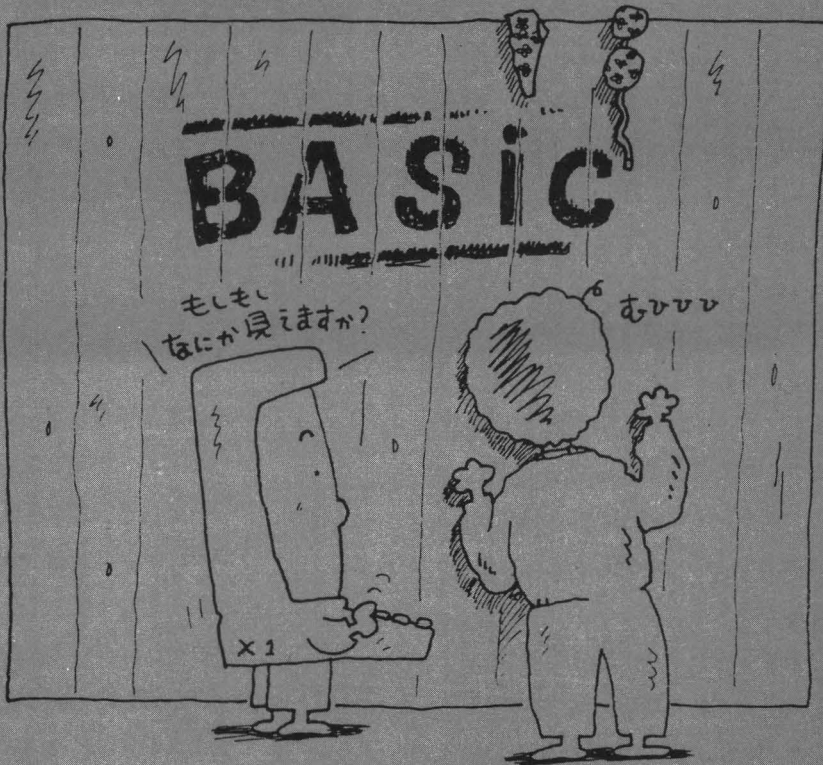
３．デジタルテロッパ（CZ-８DT）

Ｘ１の大きな特徴のひとつは，テレビ放送の画面とコンピュータで作った画像の重ね合わせができるという機能にあります。この機能をスーパーインポーズ機能といいます。スーパーインポーズの世界をさらに拡げてくれるのがデジタルテロッパ（CZ-８DT）です。

デジタルテロッパを使うと，テレビ画面とコンピュータ画面を重ね合わせたものをビデオに録画したり，ビデオカメラから送られてくる映像にコンピュータの画面を重ねて録画することができます。たとえば，自分で撮ったビデオフィルムのタイトルをコンピュータで作って重ねて録画したり，映画の字幕スーパーのように説明文を入れることもできます。デジタルテロッパとビデオを組み合わせれば，Ｘ１は，家庭の中だけではなくいろいろな業務でも幅広い用途を持つようになるでしょう。

Ｘ１の周辺機器を紹介してきました。これからは，これらすべてを含めたものを「Ｘ１システム」と呼び，本体とディスプレイの組み合わせを「基本システム」と呼ぶことにします。

2

HOW TO タッチX1！

2-1 まずはテレビ！

みなさんのＸ１のケーブルは，間違いなく接続されていますか。最初はテレビを見ることにしましょう。アンテナもしっかりとつないでありますね。では，ディスプレイテレビCZ-800Dの電源を入れましょう。

ディスプレイテレビの下側にスイッチが並んでいます（図２－１）。実は，その左側はトビラになっていて，中にこのディスプレイテレビの主電源スイッチがあるのです。ディスプレイ本体の左側面を軽く引いてトビラを開けてください。

主電源と書かれた赤いスイッチがあります。スイッチを「入」にします。右側にあるPOWER（電源）と書かれたインジケータランプが赤く点灯します。次に，ディスプレイ本体右側の電源スイッチを押してください。ディスプレイ後面の映像入力切りかえスイッチがＴＶ側になっていれば，チャンネル表示とともにテレビ放送が映るはずです。

次に，Ｘ１本体のスイッチを入れます。本体の背面にあるメイン電源スイッチをONにします。本体正面の右側の，電源スイッチもONにすると「POWER」と書かれたインジケータランプが赤く点灯します。

ディスプレイの画面は切りかわって，カセットデッキのドアが開いてしまいました。だいじょうぶ，あなたのＸ１は正常に作動しています。あわてずに，ドアを閉めてください。私たちの最初の目的は，テレビ放送を見ることでした。ゆっくりと確実に操作を覚えていきましょう。

Ｘ１のキーボードには，ずいぶんとたくさんのキーが並んでいます。これらのキーを５つのグループに分けることができます。

上の方には銀色のテレビコントロールキーが５つ並んでいます。テレビを見るためには，一番右側にあるコンピュータとテレビの切りかえスイッチを押せばよさそうです。今度はうまくいきましたね。これらのテレビコントロールスイッチと同じものは，ディスプレイテレビの方にもあります。チャンネルも音量もこのテレビコントロールキーを使ってコントロールすることができます。試してみましょう。

それでは，メインキーの左右にあるSHIFTキーと数値入力キーの3を一緒に押してください。テレビのチャンネルが３チャンネルに切りかわりました。SHIFTキーと数値入力キーの・を一緒に押してみましょう。先ほどの，テレビを映し出す以前の画面になりました。実は，この画面はコンピュータから送られているものなのです。SHIFTキーと数値入力キーの=を一緒に押すと，再びテレビ放送に切りかわります。SHIFTキーと数値入力キーの組み合わせで，さまざまな操作ができることがわかりました。表にまとめておきますから，１つ１つためしてみましょう（表２－１）。

「パソコンテレビＸ１」と呼ばれる機械(マシン)の「テレビ」の部分をまず見てみました。コンピュータ本体の電源を入れずに，テレビを単独で使うこともできます。しかし，このようにコンピュータを通じてテレビをコントロールしてみると，この機械が「パソコンテレビ」と呼ばれるのもなるほどとうなずけます。テレビに夢中になる前に，今度はパソコンの方も見てみましょう。

〔図2-1〕

フロントガラス固定ネジ

前　面

フロントガラス

コンピュータモード
表示ランプ

主電源表示ランプ

ヘッドホーン端子

電源入ー切ボタン

トビラ

音量調整ボタン

コンピュータ映像調整つまみ

順送り選局ボタン

黒レベル調整つまみ

逆送り選局ボタン

色の濃さ調整つまみ

チャンネルコールボタン

色あい調整つまみ　コンピュータ・テレビ切換えボタン

映像調整つまみ

主電源スイッチ

トビラ内操作部

背　面

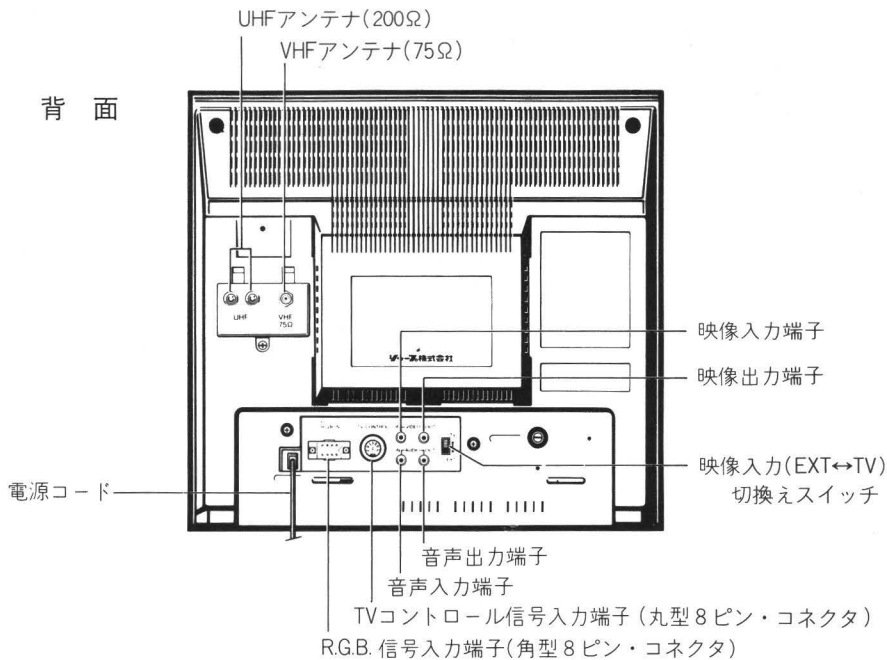
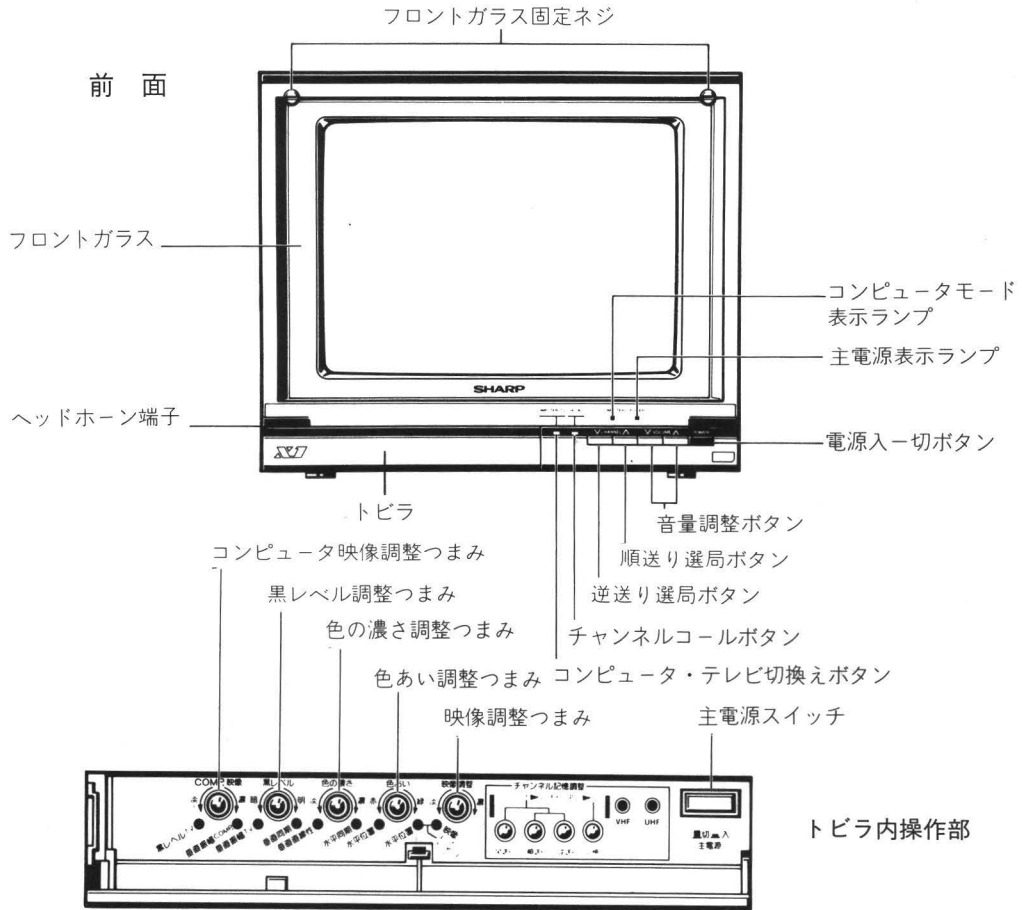



表2－1 [SHIFT] キーと数値入力キーの組み合わせ

[SHIFT] キーを押しながら [1] を押すとチャンネル1が選局できる。
[SHIFT] キーを押しながら [2]
⋮ ⋮ キーの数字に対応したチャンネルが選局できる。
[SHIFT] キーを押しながら [9]
[SHIFT] キーを押しながら [/] を押すとチャンネル10が選局できる。
[SHIFT] キーを押しながら [*] を押すとチャンネル11が選局できる。
[SHIFT] キーを押しながら [−] を押すとチャンネル12が選局できる。
[SHIFT] キーを押しながら [0] を押すと音声ミュートとなりもう一度押すと解除する。
[SHIFT] キーを押しながら [+] を押すとテレビ放送とコンピュータ画面を重ね合わせる。
[SHIFT] キーを押しながら [=] を押すとテレビ画面に切りかわる。
[SHIFT] キーを押しながら [.] を押すとコンピュータ画面に切りかわる。
[SHIFT] キーを押しながら [,] を押すと音量がノーマル位置になる。
[SHIFT] キーを押しながら [⇧] を押すと音量がアップし押し続けると最大になる。
[SHIFT] キーを押しながら [⇩] を押すと音量がダウンし押し続けると最小になる。
[SHIFT] キーを押しながら [⇨] を押すとチャンネルアップで1、2…→12と順次変化する。
[SHIFT] キーを押しながら [⇦] を押すとチャンネルダウンで12、11…→1と順次変化する。

2-2 パソコンX1スタート

●カラッポの記憶装置

ここでは，X1をパソコンとして使う方法を見ていきましょう。

X1は，メモリがすべてRAM（ラム）で構成されています。RAMというのは，「Random Access Memory」(ランダム・アクセス・メモリ）の略で，「読み出しや書き込みを自由に行なうことができるメモリ」という意味になります。すべてのメモリが，読み出しや書き込みを自由に行なえるRAMで構成されているというのは，いったいどういうことなのでしょう。

私たちは，フランス人と話をするときにはフランス語を使い，ドイツ人と話をするときにはドイツ語を使います。同じように，私たちがコンピュータと話をしながら，さまざまな指示をしたり，命令を与えたりするためには，コンピュータ専用の言葉を使います。X1では，このようなコンピュータ専用の言葉として，「BASIC」というものを用います。

「BASIC」は，「Beginner's All Purpose Symbolic Instruction Code」の略です。初心者向きに，多用途に使えるコンピュータ言語として開発されたのがこのBASICです。パソコンのコンピュータ言語としては，BASICが今日最も一般的に使われています。同じ日本語でも地方によって違いがあるように，BASICでもパソコン間での違いがあります。X1で使われるBASICは，標準的なBASICに，高度なグラフィック処理やX1独特の機能を活かすための命令をつけ加えた非常に優れたものです。このX1のBASICは，「SHARP-HuBASIC」と呼ばれます。

さて，X1ではすべてのメモリがRAMで構成されていると説明しましたが，電源を入れた直後のX1のメモリの中は空っぽになっています。人間でいえば，言葉をしゃべることのできない赤ん坊の状態です。X1と対話をするためには，つまり，X1をパソコンとして使うためには，まず，このメモリの中にコンピュータ言語を覚えさせなければなりません。RAMに対してROM（ロム）というものがあります。これは「Read Only Memory」の略で，読み出し専用メモリの意味になります。読み出し専用というからには，ROMの中にはいつでも一定の内容が収められているわけです。一般のパソコンでは，ROMの中にあらかじめBASICを覚えさせておき，電源を切ってもその中のBASICは消えずに残るようにしています。私たち人間も，一晩眠ったからといって，すっかり日本語を忘れてしまうということはありません。それではなぜX1では，電源を切るたびにメモリの中を空っぽにしてしまうのでしょうか。つまり，すべてのメモリをRAMで構成しているのでしょうか。

コンピュータ言語には，BASICの他にも「FORTRAN（フォートラン）」，「COBOL（コボル）」などさまざまなものがあります。コンピュータを利用する目的によって，これらのコンピュータ言語は使い分けられています。たとえば，FORTRANが科学技術計算用，COBOLが商業事務用という具合です。メモリの中が空っぽであれば，そのつどコンピュータ言語を覚えさせる手間はかかりますが，そのかわり，目的にあわせてメモリの中にいろいろな言語を覚えさせてコンピュータを使うことができます。私たちが，必要に応じて日本語を使ったり英語を使ったりするのに似ています。

X1では，これらのコンピュータ言語を使い分けるために，メモリの中を空っぽにしているの

です。すべてのメモリをRAMにしているのはこのためです。

●パソコンX１スタート

それではまず，X１をパソコンとして使うためにBASICをX１のメモリの中に覚え込ませましょう。

X１にBASICを覚え込ませるには，フロッピーディスク，ROM，カセットテープを用いる３つの方法をあげることができます。ここでは，X１の基本システムで使えるカセットテープを利用しましょう。

X１本体の電源を入れるとディスプレイは自動的に，コンピュータからの映像を表示するように切りかわります。ディスプレイテレビの電源が入っていなくとも，X１は自動的にスイッチをONにしてくれます。そして，X１本体のカセットデッキのドアが開き，画面２-１のようになります。

〔画面２-１〕

```
Make ready any device
Push(F,R,C or T) key
         F:Floppy
         R:ROM
         C:CMT
         T:Timer
```

X１は,電源をONにしたときにこのような動作をするように,あらかじめ命令されています。

付属のBASICテープ（CZ８CB01）をセットして（このとき，CZ８CB01と書かれた面を手前にしてください)，カセットデッキのドアを閉じましょう。すると，テープが自動的に動きはじめます。X１がメモリの中にBASICを読み込んでいるのです。

フロッピーディスクやROMがセットされている場合，X１はフロッピー，ROM，テープの順にBASICの読み込み先をさがしていき，接続されていることが最初に分かった周辺機器からBASICの読み込みを開始します。

BASICの読み込みが終わると，テープは自動的に巻き戻され画面２-２のようになります。

〔画面２-２〕

```
  SHARP-HuBASIC CZ-8CB01 V1.0
Copyright (C) 1982 by SHARP/Hudson

23536 Bytes free
Ok
```

X 1を，BASICの命令で扱うことができるようになりました。ここから先は，みなさんからの命令でX 1のコントロールをしなければなりません。

＊BASICマニュアルの195ページに，BASICテープの複製(コピー)を作る方法が書かれています。この方法に従ってBASICテープのコピーを作り，ふだんはコピーを使うようおすすめします。強い磁気にテープを触れさせるなどの誤った扱いをして，BASICテープの中身が壊れてしまったときやテープがゆがんでしまった場合でも，もとのテープが保管してあれば安心です。

らんだむめも

メモリ（memory）とは

コンピュータ関係の本を見ると，よく出てくるなじみの深い言葉です。日本語に直せば，「記憶」あるいは「記憶装置」となります。コンピュータの持っている情報や，あなたの入力した情報を格納する，コンピュータには不可欠な要素がメモリです。

コンピュータの働きの中心となるCPU(中央演算処理装置)は，一度に多くの情報を扱うことができません。そこで，メモリに情報を格納しておき，必要なときに読み出して処理をします。

電話帳を例にあげてみましょう。電話帳にはたくさんの情報(電話番号)がつまっています。この電話帳の全体をメモリと考えてください。あなたが電話帳を開いて必要な電話番号を調べるように，コンピュータはメモリから必要な情報をさがして読み出し，その情報を利用します。

メモリにはByte単位で番地がつけられています。電話帳のページのようなものですが，メモリでは1つの番地に1つの情報だけが格納されます。

メモリにRAMとROMがあることは，ご存じのとおりです。X 1では，BASICでもその他の高級言語のシステム・プログラムでも，すべてRAMに書き込まれます。X 1のROMにはIPLといって，BASICなどのシステム・プログラムを外部記憶装置から読み込んで，RAMに書き込む仕事をするプログラムが格納されています。

2-3 BASICの世界へ第一歩！

●アプリケーションテープをのぞきみ

BASICの読み込みを終えたX１は，あなたからの命令を待っています。

さっそくいろいろな仕事をさせてみたいところですが，まず最初は，付属のアプリケーションテープを使ってみましょう。

アプリケーションテープを本体のカセットデッキにセットし，[SHIFT]キーと[F 1]キーとを一緒に押します。[F 1]から[F 5]のキーを，**ファンクションキー**と呼びます。

それぞれのファンクションキーには，X１をコントロールするための命令が定義されています。[SHIFT]＋[F 1]は[F 6]を表わし，[SHIFT]＋[F 5]は[F 10]を表わします。ですから，この５つのファンクションキーに，全部で10の命令が収められていることになります。

画面にOKと表示されましたか？　OKと表示されたら，今度は[F 5]キーを押してください。いかがですか，画面にはX１の美しいロゴマークが現われ，同時に楽しい音楽が流れます。

ここでファンクションキーの機能について少し勉強しておきましょう。

[F 6]には，LOAD(ロード)＋⏎という命令が定義されています。LOADというのは，テープからプログラムを読み込みなさいという命令です。⏎はリターンキーを表わします。[F 5]キーには，RUN(ラン)＋⏎という命令が定義されており，RUNはプログラムを実行しなさいという命令です。X１はLOADやRUNなどの命令を受け，リターンキーを押されたときにはじめて，その命令の動作をします。ところが便利なことに，[F 5]キーにも[F 6]キーにも，あらかじめリターンキーを押すのと同じ内容が定義されていますから，あらためてリターンキーを押す必要はありません。

表２－２にBASICテープをCZ８CB01で設定してあるファンクションキーの内容を示しておきます。

KEY LIST ⏎

と入力してやると，画面に表２－２と同じ内容が表示されます。どこに何が入っているか覚えてしまえば便利ですが，忘れてしまったらこうしてファンクションキーの内容を呼び出してみてください。また，ラベルをはっておくのもよい方法です(ラベルのはり方は，マニュアルを参考にしてください)。

表２－２　ファンクションキーの内容

ファンクションキー	設定内容	ファンクションキー	設定内容
F 1	AUTO	F 6	LOAD⏎
F 2	?TIME$⏎	F 7	WIDTH
F 3	KEY	F 8	CHR$
F 4	LIST⏎	F 9	PALET
F 5	RUN⏎	F10	CONT⏎

また，BASICテープで設定されていない命令で，あなたがひんぱんに使う命令があれば，それをあらたにファンクションキーの内容に入れることもできます。

たとえば，[F 1]にPRINTという命令を入れておきたいというときには，次のようにします。

1．KEY LIST ⏎でキーの内容をリストする。

2．カーソルを〝AUTO〟のAのところまで移動させる。

3．[SHIFT]キーを押しながらPRINTと入力する。

4．SHIFT キーを押しながら Ⅱ を押し，スペースキー
　でCHR$（13）を消す。
5．↵ を押す
これで，F 1 にPRINTという命令が入力されました。
それでは，SHIFT キー＋ F 1 キー，F 5 キーを使って，アプリケーションテープの中身をのぞいてみましょう。アプリケーションテープの中身を見るとＸ１の特徴的な機能もよくわかります。

1．TITLE　X I
最初に画面に表示されたＸ１のロゴマークは，Ｘ１の特徴的な機能の１つである，**プログラマブル・キャラクタ・ジェネレータ**（PCG）によって描かれています。この時に流れる音楽も，８オクターブ３重和音を発生する**シンセサイザ機能**によって演奏されています。画面が止まりOKと表示されたら，SHIFT キー＋ F 1 キーを使って次のプログラムを読み込ませ，F 5 キーで実行させます。

2．TEST　X I
Ｘ１の各機能をチェックする働きを，このプログラムは持っています。万一故障かと思われるようなことが起きたら，このプログラムを動かしてみるとよいでしょう。

3．TRAIN　X I
PCGで描かれた新幹線が，グラフィックで描かれた電柱の間を走ります。新幹線は，こちらの電柱と向こうの電柱の間を走っているように見えます。Ｘ１の特徴的な機能である**プライオリティ機能**（優先順位機能）によるものです。また，Ｘ１の**時計機能**によって現在時刻を表示しています。

4．3D DEMO　X I
グラフィック処理によって円板を積み重ねたような形のコマを描き，これもＸ１の優れた機能である**タイリングペイント**によって，明暗をつけて塗り分けています。タイリングペイントとは，赤，緑，青の３色を１ドット単位でペイントできるもので，この機能を使うことで中間色や縞模様を出すことができます。たとえば，黄緑色は黄色と緑色の１ドット単位で交互に点灯させればよいわけです。さらに，**パレット機能**（図形や文字の色を一緒に変える機能）というものを利用し，あたかもコマが回転しているように見せています。これについては，後の項で詳しく説明します。

5．3D GRAPH　X I
立体棒グラフを作ります。また，PATERN文を使って漢字の表示も行なっています。ビジネスプログラムを作ろうとする方は，このプログラムが応用できるでしょう。

6．BINGO　X I

これはゲームプログラムです。X 1の機能を活用した美しい画面を持っています。遊び方は，このアプリケーションテープに付属している説明書を読んでください。

BASICの世界への第一歩を進めた私たちは，このようなプログラムがどのように作られているのかを，これから勉強していきます。X 1の特徴ある機能についても，徐々に勉強していきましょう。それは驚くほど奥深い世界ですが，それだけに知る楽しみも大きく，きっとみなさんを夢中にさせることでしょう。

「今すぐX 1のパソコン機能を活用したい」という方は，市販されているパッケージソフトを使うこともできます。すでに私たちは，パッケージソフトをX 1に読み込ませる方法も，読み込ませたプログラムを実行させる方法も勉強しました。パッケージソフトを動かしながらBASICの勉強を進めていくこともできますし，また，それらのプログラムの内容は，これからの勉強に大変参考になりますから，積極的に活用してください。

2-4 スクリーンのらくらく編集

●カーソルを思いのまま動かす法

実際にプログラムを自分で作ったり，雑誌などに出ているプログラムを自分で試してみる場合には，画面を見ながら１つ１つのキーを押して入力していきます。慣れないとなかなか大変ですし，いくら慣れた人でも打ち間違えることがあります。プログラムの修正をしなければならないこともありますし，部分的な削除を行なったりすることもあります。このような作業を「画面の編集作業」といいます。画面の編集は，すべてカーソルによって行ないます。カーソルというのは画面上で点滅している小さな四角形のことです。これは編集作業をするあなたの有能な編集助手。画面中を飛び回って，あなたの命令どおりに動きます。

さっそく，カーソルをあなたの思いのままに動かしてみましょう。

キーボードの右のテンキーの下に矢印のついたカーソルコントロールキー[⇨]がありますので，押してみましょう。カーソルが１文字分右に移動します。ずっと押し続けると走るようにして動いていきます。右端までくると，カーソルは１行下がって左端に現われます。[⇦]は，カーソルを左に移動させるキーです。[⇧]は，カーソルを上に移動させることはわかりますが，[⇦][⇨]のキーと異なっていくら押し続けても，最上行まで移動したらカーソルは止まってしまいます。[⇩]は，カーソルを下に移動させます。カーソルが最下行までいったとき，さらに押し続けると画面全体が上に流れていきます。このように画面が流れていくことを**スクロール**といいます。

カーソルキーだけに限りませんが，同じキーを押し続けると，画面上を途中で休むことなく，画面の端まできても止まることもなく，ずっと動き続けますね。

この機能を**リピート機能**といいます。

REPEAT OFF

と入力しリターンキーを押してみましょう。この機能が消えて，いくらキーを押し続けても１文字分しかカーソルは動きません。一度押して１文字分だけ移動，というのでは面倒ですから，元にもどしましょう。

REPEAT ON

としてリターンキーを押します。テンキーの上にある[CLR HOME]キーを押すと，カーソルは画面の左上角にパッと飛んでいきます。ここがカーソルの**ホームポジション**です。[SHIFT]キーと[CLR HOME]キーを同時に押すと，画面に書いてある文字がすべて消えて，カーソルはホームポジションに移動します。

●文字の挿入と削除

カーソルの移動よりもずっと大切な作業が，文字を挿入したり，削除したりする編集作業です。

まず，文字の挿入をしてみましょう。画面にあなたの苗字と年齢を書いてください。苗字と年齢の間に名前を書き入れましょう。年齢のところまでカーソルを移動していき，そこで[SHIFT]

キーと[INS DEL]キーを同時に押します。1文字分の空白ができますね。そうしたら，[SHIFT]キーを押しながら[INS DEL]キーを挿入したい文字数だけ押します。

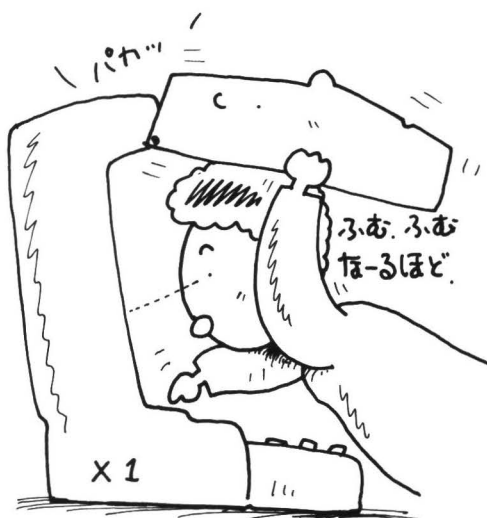
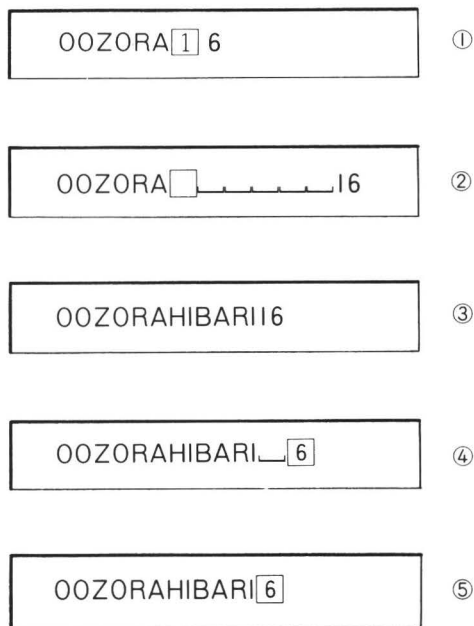
たとえば，

OOZORA16

と文字があるとき，Aと1の間に「HIBARI」と文字を入れてみましょう。カーソルを1のところにあわせます（図2－2①）。そこで[SHIFT]＋[INS DEL]を6回押します。すると，カーソルから右に6文字分空白ができますので（同②），キーを押してAと1の間にHIBARIと入れることができます（同③）。

〔図2－2〕

OOZORA[1]6	①
OOZORA[ ]______16	②
OOZORAHIBARI16	③
OOZORAHIBARI_[6]	④
OOZORAHIBARI[6]	⑤

文字の削除を行なう場合は，スペースキー（横に長く一番大きなキーで，スペースバーとも呼びます）を使う方法や[INS DEL]キーと[E]キーを合わせて使う方法，[CTRL]キーを使う方法があります。先ほどの，OOZORAHIBARI16の16を例にとると，カーソルを1のところに移動して，スペースキーを押していきます。しかし，スペースキーを使って文字を消すと，そこに空白ができてしまいます。たとえば，1を消しますと，そこに1字分のすき間ができます（同④）。この場合，[INS DEL]キーを使うと，文字を削除すると同時にその右側の文字をつめることができるのです。カーソルを6のところに移動し，[INS DEL]キーを1回押します。すると，カーソルの左の空白が削られます（同⑤）。カーソルの右側に文字があれば，その全体が左に移動します。キーを押し続けるとカーソルは文字を削りながら左端までいき，今度はカーソルの右の文字だけが左に移動して，ちょうどカーソルに吸い込まれるように左端から削られ，最後にはすべてを削除してしまいます。

● [CTRL] キーはスクリーンエディットのスーパースター

文字を削除することは[CTRL]キーを使ってもできます。[CTRL]＋[E]と押すと，その行のカーソルから右がすべて消えてしまいます。また，[CTRL]＋[Z]と押すと，カーソルの右からその下にある文字をすべて消すことができます。[CTRL]キーはその他たくさんのスクリーンエディ

ット機能を持っています。これを表2-3に整理してみました。

プログラムの作成というのは，入力し，実行してエラーを発見し，さらに，スクリーンエディタによって修正を行なうという作業の繰り返しです。スクリーンエディット機能を一度に覚えていくのは大変ですが，これらの作業を通して自然に覚えていきますから，あまり心配せずに，できるだけキーボードに触れるようにしましょう。

表2-3 CTRL キーのスクリーンエディット機能

キー	機能
CTRL + F	カーソルが1ワード分ずつ右へ移動。
CTRL + B	カーソルが1ワード分ずつ左へ移動。
CTRL + T	水平タビュレーションを設定。
CTRL + I	CTRL + T で設定した位置にカーソルを飛ばす。
CTRL + Y	CTRL + T で設定した位置でこのキーを押すと，その位置だけ水平タビュレーションが解除されます。
CTRL + W	このキーを押した行と次の行をつなげる。
CTRL + J	カーソルのある位置から右の文字をラインフィードして次の行に移す。
CTRL + N	カーソルのある行から上の文字を上方向にスクロールする。
CTRL + O	カーソルのある行から下の文字を下方向にスクロールする。
CTRL + M	リターンキーと同じ。

2-5 カセットテープのコントロール

●カセットテープ活用法

本体にカセットデッキが内蔵されているというのもX1の大きな特徴です。このカセットデッキを活用できるように，いろいろなしくみや命令が用意されています。

X1付属アプリケーションテープのように，いくつかのプログラムが記録されているテープの内容を知りたい場合は，[F][I][L][E][S]とキーインして，リターンキーを押します。テープを最初まで巻き戻した後に，テープを送りながらプログラム名を次々に表示していきます。試してみましょう。

目的のプログラムが見つかったら，キーボードの右側にある数字キーの上にあるカセットコントロールキーを使ってテープを止めます。カセットコントロールキーには巻き戻し〔◀◀〕，早送り〔▶▶〕，停止〔■〕の3つがあります。次に[A][P][S][S][−][1]とキーインしてみましょう。キーインしたらリターンキーを押します。[SHIFT]キーと〔◀◀〕キーを一緒に押しても同じです。この操作で，目的のプログラムが記録されている最初の部分まで，テープを巻き戻してくれます。ここでLOAD命令とRUN命令を使えばプログラムが実行されることは，もうご存じのとおりです。ここでは，別の方法を試してみましょう。

[R][U][N]["][C][A][S][:]["]とキーインしてみてください。プログラムを読み込んだ後，X1は自動的にプログラムの実行をはじめます。

もし，プログラムの名前が最初からわかっているときには，[L][O][A][D]["]プログラム名["]とキーインします。X1は，テープを早送りしながら目的のプログラムをさがし，見つけたところで自動的に読み込みをはじめます。

RUN"CAS：プログラム名"とすると，テープを早送りしながら目的のプログラムをさがしてくれます。見つけたところで自動的にロードした後，X1は実行もしてくれるのです。

●[SHIFT]キーでテープを操作

もう一度，カセットコントロールキーを見てみましょう。この3つのカセットコントロールキーは，[SHIFT]キーと合わせることで6つの働きを行ないます。

〔◀◀〕キーはテープの巻き戻しをします。BASICでREWという命令を与えたときにも同じ働きをします。試しに[R][E][W]とキーインしてリターンキーを押してみてください。次に[SHIFT]キーと〔◀◀〕キーを一緒に押してみましょう。こうすると，[A][P][S][S][−][1]と同じく，プログラムの最初の部分までテープが巻き戻されます。

〔■〕キーはテープの走行をストップします。[SHIFT]キーと一緒に押すと，カセットデッキのドアが開きます。EJECTという命令でも同じ動作をします。〔▶▶〕キーと[SHIFT]キーを一緒に押すと，[A][P][S][S][+][1]と同じく，次のプログラムの最初の部分まで早送りします。なかなか複雑ですから表にまとめておきましょう(表2-4)。

表２-４　カセットテープのコントロール

	SHIFT なし	SHIFT あり
◀◀	REW カセットテープの巻き戻し命令	APSS－1 前のプログラムの頭出し
■	STOP カセットテープの走行停止命令	EJECT カセットデッキのドアを開く
▶▶	FAST カセットテープの早送り命令	APSS＋1 次のプログラムの頭出し

●CMTでテープを回せ

CMTという命令文もあります。これもカセットテープをコントロールするための命令です。命令の後に書く数字によっていろいろな働きをX１にさせることができます。これも表にまとめてみましたので，試してみましょう（表２-５）。

CMT=10とすると，カセットの中のプログラムを消去します。カセットテープの爪が折ってあれば，間違えて消してしまうこともありませんから，プログラムを入れたカセットテープの爪は必ず折っておくべきです。

表２－５　CMTによるカセットコントロール

CMT=0	EJECT	カセットデッキのドアを開く。
CMT=1	CSTOP	走行停止
CMT=2	PLAY	再生
CMT=3	FAST	早送り
CMT=4	REW	早戻し
CMT=5	APSS+1	頭出し（順方向）
CMT=6	APSS－1	頭出し（逆方向）
CMT=10	RECORD	録音，SAVE

ここまで見てきたように，X１のカセットデッキはすべてキーボードから操作することができます。そしてBASICの中にもREWやFASTのようにさまざまなカセットコントロールの命令文があるのです。最後にAPSSという命令文についてだけ補足しておきます。

APSSの後には，+50から－50までの数字を書くことができます。APSS＋３とすれば，プログラムを２つ飛び越えて３つ目にあるプログラムの頭出しをします。いきなりカセットテープのコントロールの説明をされても…と思う人もいるかもしれませんが，これらの機能はX１ならではのもので，大変便利に使うことができます。プログラム作りが進むにつれて，これらのカセットコントロール命令が活躍するようになりますから，ぜひ覚えてください。

2-6 X1のもうひとつの機能

●パソコン時計X１？

パソコンテレビX１が，テレビとパソコンの２つの機能を持っていることは，今さらあらためていうまでもありません。X１は，この２つの機能をいっそう活躍させるために**時計機能**も持っています。時計機能というのは，**クロック機能**と**タイマー機能**のことです。クロック機能もタイマー機能もX１の本体に内蔵された電池によってバックアップされています。コンピュータやディスプレイの電源を切っても，この機能は電池のおかげで働き続けているということです。この電池は，自動的に充電するタイプのものですから交換の必要はありません。

ではこれらの機能を呼び出してみましょう。

X１にBASICを読み込ませる前であれば，Ⓣキーを押すだけで呼び出すことができます。BASICをロードにした後であれば，ⒶⓈⓀとキーインしてリターンキーを押します。画面は次のようになります。

〔画面２－３〕

```
TV Timer control

83/06/14 WED 10:34:54

TIMER1   XX/XX XXX XX:XX    OFF
TIMER2   XX/XX XXX XX:XX    OFF
TIMER3   XX/XX XXX XX:XX    OFF
TIMER4   XX/XX XXX XX:XX    OFF
TIMER5   XX/XX XXX XX:XX    OFF
TIMER6   XX/XX XXX XX:XX    OFF
TIMER7   XX/XX XXX XX:XX    OFF

           Month  01 - 12 or XX
[ESC]=Exit, [CLR]=Reset, [CR]=Set
```

現在の年/月/日/曜日，時：分：秒が表示されています。これがクロック機能です。あなたのX１のクロック表示は合っていますか？　合っていなければ，正しく設定し直しましょう。

０から９までの数値入力キーを一般にテンキーと呼びます。テンキーの下の矢印が書かれた４つのキーをカーソルキーと呼ぶことはもうご存じですね。

それでは，カーソルキーを使って，カーソルをクロック表示の左端に移動させましょう。年/月/日/曜日，時：分：秒の順にキーインしていきます。時刻は24時間制で表示しています。時

刻を正確に合わせるには，たとえば午後3時23分を少し過ぎたところであれば，クロック表示を，15：24：00としておき，電話の時報などを聞きながら24分ちょうどになった瞬間にリターンキーを押します。これで秒まで正確に合わせることができました。

●タイマーをセット

次に**タイマー機能**を見てみましょう。タイマー機能は，テレビ放送の「ON/OFF」を設定するものです。特定の月日のある時刻を指定したり，毎日決まった時刻を指定してテレビの「ON/OFF」をコントロールしたりできます。このタイマー機能によってテレビがONになった場合，コンピュータを使用中でも，画面は自動的にテレビ放送に切りかわります。このタイマー機能を利用すれば，プログラム作りやゲームに夢中になっていても，X1は忘れずにテレビ放送に切りかえてくれます。

タイマーの設定は，カーソルの位置に従って「月/日/曜日，時：分，ON/OFF」を書き込むだけです。リターンキーを押すと，タイマーがセットされます。一度セットしたタイマーを解除するには，SHIFTキーとCLRキーまたは，CLRLキーとLキーを一緒に押します。

内蔵時計にセットされている年月日，時刻，曜日はBASICからDATE$，TIME$，DAY$という関数で呼びだして利用することができます。また，作ったプログラムをテープなどに記録すると，年月日，時刻，曜日も自動的に記録されます。これらの日付けなどは，FILESによって見ることができます。このときに記録されている内容の種類も表示されます。Basと表示されるのはBASICのプログラム，Ascはアスキー形式で記録されたプログラムやデータ，Binは機械語で記録されたプログラムやデータを示します。

2-7 タイマー機能でテレビをコントロール！

●BASICの命令でテレビをコントロール

キーボードからテレビのチャンネルや音量を直接コントロールする方法を見てきましたから，今度は，BASICの命令によってテレビをコントロールしてみましょう。

スペースキーの横に[CAPS LOCK]（キャプスロック）キーがあります。このキーを押すと，画面には大文字だけが表示されるようになります。小文字を表示させるときには，[SHIFT]キーと文字キーを一緒に押します。

それでは，キーボードから次のように打ち込んでください。

TVPW ON⏎

画面が切りかわり，テレビ放送を映すようになりました。TVPWは，テレビの電源の「ON/OFF」と指示する命令文です。さてパソコンからの映像は見えなくなってしまいましたが，X１では，テレビとパソコンの両方の画像を重ね合わせて表示することもできるのです。試してみましょう。[SHIFT]キーとテンキーの[+]を一緒に押せばよいのですが，ここではBASICの命令文を使って，同じことをX１に実行させてみましょう。

画面が見えませんから十分に注意して，次のように打ち込みます。

[C][R][T]　[2]⏎

テレビ放送の画面に重なって，今打ち込んだ文字が表示されます。CRTは，ディスプレイテレビをコントロールするためのBASICの命令文です。この命令文は，《２-５》で学んだCMT命令と同じように，後に書く数字によってさまざまな働きをします(表２-６)。

表２-６　CRT命令による画面表示

	n	画面表示
CRT	0	テレビ放送を表示。
	1	コンピュータ画面を表示。
	2	テレビ放送のコントラストを下げて，コンピュータ画面を重ねて表示。
	3	テレビ放送とコンピュータ画面を同時に重ねて表示。

CRT命令では，後に書く数字を省略してはいけません。なぜこんなことをわざわざ注意するのかというと，X１のBASICには，後に書く数字を省略してもよい場合があるからです。正確にいえば，省略してもBASICをロードした直後の標準的な状態へ自動的にセットされるだけの命令文があります。VOLという命令は，音量を変える命令で，この１つの例です。後に書く数字を省略すると，標準的な音量になります。CRT命令では，後に書く数字を省略すると誤り(エラー)になってしまいます。

このように，命令文の後に書いて命令の内容を指示する数字をオペランド(演算数)といいます。

さて，CRT２⏎とキーインすることで，画面にテレビ放送が映し出されていても，パソコンX１はあなたからの命令を待っていることも分かりました。BASICテープをロードしている間やテレビ放送を見ながら待つこともできます。

だんだんとパソコンテレビＸ１の優れた機能がわかってきたことでしょう。Ｘ１の特徴を活かして，テレビをコンピュータでコントロールする「テレビ24時間予約プログラム」を作るための下準備はこれで整いました。

プログラムの第１歩は〈ゆっくりと確実に学んでいくこと〉。そうすれば，Ｘ１もBASICもプログラム作りのテクニックも，必ずみなさん自身のものになりますから，ここはひとつ腰をすえてじっくり取り組んでください。

らんだむめも

算術演算子

Ｘ１では，計算に使う記号（算術演算子）は次の表のようになります。

算術演算子	演算内容	優先順位	例	普通の計算
^	累乗演算	1	X^3， X^Y	(X^3， X^Y)
＊	乗　算(×)	2	A＊3， A＊B	(A×3, A×B)
／	除　算(÷)	2	A/12, A/B(ただし，B≠0)	(A÷12, A÷B)
−	マイナス符号	3	−A	
＋	加　算(＋)	4	A＋B	
−	減　算(−)	4	B−B　　A−B	
¥	整数除算	5	10¥3	
MOD	剰余演算	5	32.8MOD7.25	

＋－はそのままですね。×は＊に，÷は／になります。累乗演算のときは，^の後にベキ乗数を書きます。たとえば16^2だったら16^2と表わします。また，変わった演算子で¥とMODがあります。¥は整数どおしの割り算をします。実数は，整数に計算（四捨五入されて）され，答の整数部（小数点以下は切り捨てられます）を結果として表示します。

16.6¥3.2→17¥3＝5

MODは¥と同様な計算をしますが，¥と違って整数で割ったときの余りを表示します。

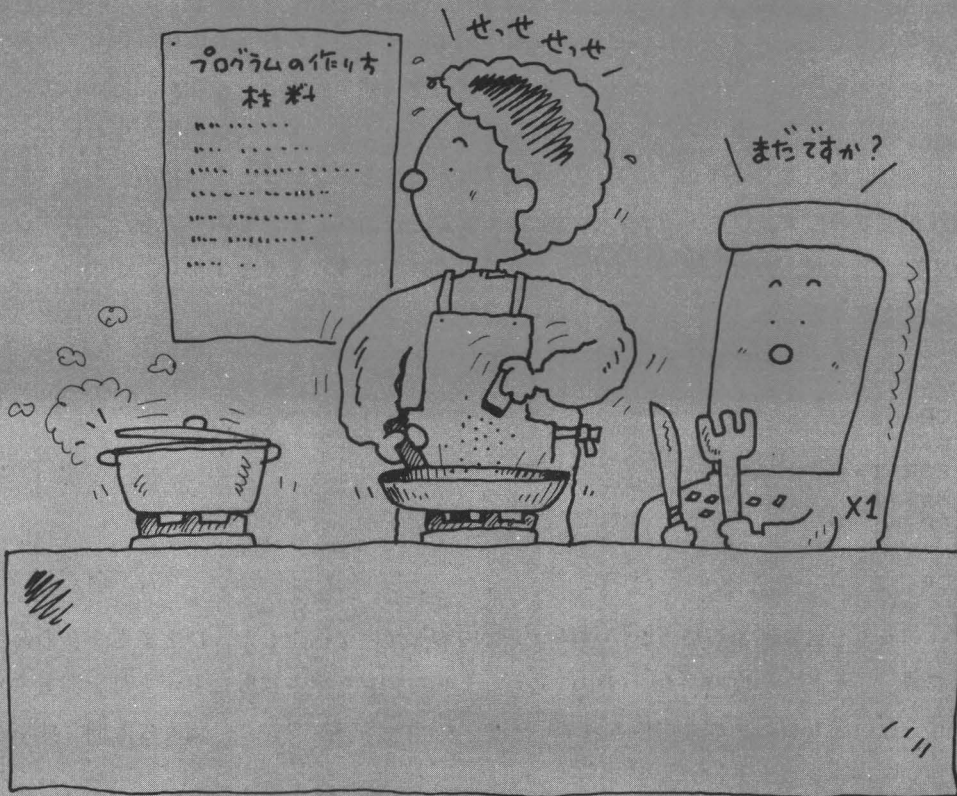
16.6MOD3.2→17MOD 3＝2（17÷3＝5……余り2）

（5：¥の結果　2：MODの結果）

この２つの演算子は使い方仕第ではとても役に立ちます。

表に優先順位という欄がありますが，計算式を書いた際、この優先順位の高い（数字の小さい）順に計算されます。優先順位が等しい場合は左から順に，また( )で囲むと，その中の演算を先に行ないます。

3

テレビ24時間予約プログラム

3-1 プログラムとはこんなもの

●プログラムとはこんなもの

まず次のように打ち込んでみてください。

10 CRT 0

これまでと違って，リターンキーを押しても何も起こりません。

前に「CRT 0」とキーインしたときには，リターンキーを押すとディスプレイの画面がテレビ放送に切りかわりました。ところが，ここでは何も起こりません。

この前についている，10という数字に秘密がありそうです。

これまで学習してきたところでは，キーボードから命令を打ち込んでリターンキーを押すと，直ちにその結果が画面に現われました。直ちに結果が現われる使い方を「ダイレクトモード」と呼びます。それでは，すぐには結果の現われない「10 CRT 0」というのは，いったい何なのでしょうか。

実は，これがプログラムというものです。先頭にある数字は，**行番号**（ラインナンバー）と呼ばれるもので，パソコンに仕事をさせる順序を示します。この行番号の後に命令文を書き，リターンキーを押していくと，パソコンのメモリの中に仕事の内容や手順が記憶されていきます。この操作をプログラミングとかコーディングといいます。すぐには結果が現われないので少しつまらない気もしますが，この「プログラム」にこそコンピュータの特徴である，無限の可能性があるのです。プログラムの仕方を覚えることが面倒で退屈であっても，プログラムによって必ずＸ１が，あなたの手足となって働いてくれるようになりますから，頑張ってじっくり覚えてください。

では，さっそく次のプログラムを打ち込んでみてください。

```
10 CRT 0
20 S=0
30 FOR I=1 TO 4444
40 S=S+I
50 PRINT S
60 NEXT I
70 CRT 2
```

プログラムとは，簡単にいうと，パソコンに内容と順序を指定した，ひとまとまりの命令のことです。しかし，一度パソコンの中に覚えこませたプログラムは私たちがそれを消去する命令を与えない限りいつまでも記憶されています。ですからパソコンに，何度でも同じ仕事を繰り返して実行させることができます。

それでは今記憶させたプログラムを実行させてみましょう。プログラムを実行させる命令はRUNです。

プログラムを実行すると，ディスプレイはいきなりテレビ画面に切りかわりました。しばらくすると，テレビ放送の画面に重なって「9876790」という数字が表示されます。これは，１から4444までの整数を全部足し算した答です。パソコンが，この足し算をしている間，テレビを

楽しんでいただいたというわけです。

これは，必要に応じてテレビ画面を選択したり，コンピュータの画面を映し出したり，あるいはその両方を重ねあわせたりすることができるという，X１の機能をBASICでのプログラムでコントロールできるのだという実例です。プログラムの内容は順次理解していけばよいでしょう。ところで，いま，あなたのコンピュータ画面はテレビ放送と重なったままですね。SHIFT ＋ ・（これはテンキーについているほうを使います）で元のコンピュータ画面に戻すことができます。

●コマンドとステートメントの違いは？

プログラムの勉強に本格的に入る前に，コンピュータとの対話に大切な命令語についてちょっと知っておいてほしいことがあります。今までRUN，LOADなどの命令がでてきましたね。これを「**コマンド**（COMMAND)」といいます。日本語に訳すと「命令」という意味になりますが，行番号をつけてプログラムの中に書き入れることは原則としてできません。一般に，外からプログラムに働きかけをする命令文です。

これに対して，PRINTやNEXTのようにプログラムの中に行番号をつけて書き込むものを「**ステートメント**（STATEMENT）といいます。日本語に訳すと「記述」という意味になり，行番号をつけることによって，さまざまな仕事をまとめてコンピュータに命じることができます。RUN命令で走らせてコンピュータに仕事をさせる状態を「ステートメントレベル」，コマンドを使う状態を「コマンドレベル」と呼びます。コマンドとステートメントの違いを簡単に書き表わすと次のようになります。

命令文
- ステートメント（STATEMENT）……PRINT文やEND文のように，行番号をつけてプログラムの中へ書き込む命令文。ダイレクトモードで使えるステートメントもある。
- コマンド（COMMAND）……………原則として，行番号をつけてプログラムの中へ書き込むことはできない。RUN，LIST，NEWのように外からプログラムへ働きかけをする。

●プログラムと材料

今度は，テレビのチャンネルをBASICのプログラムでコントロールしてみましょう。まず，先ほど入力したプログラムをX１のメモリから消去します。

NEW

とキーインしてください。

NEWは，「新しい仕事に切りかえるぞ」という命令です。つまり，NEWは，現在記憶されているプログラムを消去してしまう命令です。プログラムが消去され，X１は次の命令を待っています。それでは，次のプログラムを打ち込んでください。

文字を打ち間違えたら，カーソルキーを上手に使って修正します。表示された文字の位置にカーソルを移動し，１行打ち込んだら忘れずにリターンキーを押してください。リターンキーを押さないで，そのまま続けて次の命令を打ち込んでしまうと，コンピュータは前に打ち込ん

だ命令と次に打ち込んだ命令の区切りがわかりません。区切りのサインをコンピュータに送ってやらねばなりません。これがリターンキーというわけです。

```
10 INPUT CH
20 CRT 0
30 CHANNEL CH
40 GOTO 10
```

プログラムを実行すると，「？」マークが表示されます。テンキーで希望するチャンネルの数を入力してリターンキーを押すと，画面はそのチャンネルのテレビ放送に切りかわります。他の数字キーを押してもう一度リターンキーを押してみてください。間違いなくチャンネルが切りかわりました。

これまで見てきたように，リターンキーを押すまでコンピュータは反応しません。リターンキーはプログラムの行の終わりや，データの終了合図をコンピュータに送る役割をもっていることがこれでわかりました。と同時に，プログラムを使うとダイレクトモードとは異なって，一度入力すれば何度でも繰り返し同じ仕事をさせることができるのもわかりました。

それでは，このプログラムの中身をじっくりと見てみましょう。

行番号10のところ（10行と呼びます）のINPUT文は，パソコンが人間からのデータの入力を必要とするときに使う命令です。データというのは，材料のことです。

料理のことを思い浮かべてみましょう。最初に新鮮な材料をそろえなければなりません。

次にその材料を加工し，手順に従って調理をします。プログラムが料理の手順で，材料の加工はプログラムの中でのデータ処理にあたります。

このプログラムでは，コンピュータがチャンネルを選択するのではなく，人間の希望するチャンネルを入力するようになっています。入力を待ち構えるのがINPUT文です。料理でいえば，ここで材料を鍋に入れるということになります。プログラムを実行すると，X１は「？」マークを表示してデータを入れてくれと要求してきます。

INPUT文は，

INPUT　変数

という形で使います。このプログラムでは希望するチャンネルをCHという変数に入れています。いきなり変数という言葉を持ち出してしまいましたが，先に進む前に，どのようなプログラムにも欠かすことのできない変数というものを調べておきましょう。

●変数ってなんだ？

変数というのは，プログラムの実行によって変化する数であり，何かの値を収めておく箱です。

変数の反対は定数です。定数というのは，文字どおり定まった数のことです。プログラムを実行しても，その中で使われている定数は常に一定です。RUN命令でプログラムを実行したとたんに４が５になったり，５が６になったりするようなことはありません。

変数は，これに対して自由に変えられる数なのです。ですから数そのものとして考えるよりも，数の記憶場所，データを収めておく箱と考えた方が正確なのです。

さて，この変数には２つの種類があります。**数値変数**と**文字列変数**です。数値変数という箱

には，数値しか入りません。数値というのは，そのまま四則の計算ができるもので，言いかえれば，量を表わす数のことです。文字列変数には文字列だけが入ります。文字列というのは住所や名前などの文字の並びです。

BASICにはLET文というものがあります。次のプログラムを見てください。

```
10 LET A=4
20 B=3
30 C=A+B
40 PRINT C
```

ここで使われている「A」，「B」，「C」が数値変数です。LET文は，右辺を左辺に代入しろという指示をするステートメントです。ですから「A＝3」は「Aは3に等しい」という意味にはなりません。Aが3に等しいのであれば，「3＝A」と書いてもよさそうですが，BASICでは，このように書けば，エラー（誤り）となります。20行と30行ではLET文を省略していますが，20行は変数Bに3を代入しなさいということであり，30行は変数Cに変数Aの値と変数Bの値の和を代入しなさいということを意味しています。なお，LETは完全省略ステートメントなのでプログラムの中に書かなくてもかまいません。40行のPRINT文は，Cという変数の中身を画面に表示しなさいという意味になります。このプログラムを実行すると，画面には「C」という文字ではなく，「7」という数値が表示されます。AもBもCも数値変数であること，つまり数値を入れるための箱であることを，もう一度思い出してください。

それでは，別のプログラムをもう1つ見てください。

```
10 A$="ﾊﾟｿｺﾝ"
20 B$="TV X1"
30 C$=A$+B$
40 PRINT C$
```

前のプログラムと異なる点がいくつかあります。ここで使われる「A$」，「B$」，「C$」が文字変数なのです。文字変数は，文字を入れるための専用の箱です。「$」（ダラーマーク）をつけて，数値変数ではないことを表わしています。LETは，右辺を左辺に代入しなさいという指示であることはもうわかりました。それでは，右辺の「”」は何なのでしょうか。数値と文字列を区別するために，このクォーテーションマークを使うのです。たとえば，数字も文字ですから文字列として使うような場合もあります。数値というのは量を表わす数のことですから，電話番号のようなものは数値ではなく，数字の文字列といえます。「135」は数値ですが，これを「”」（ダブルクォーテーション）で囲って「“135”」としてやれば文字列として扱うことができるようになります。試しに，前のプログラムの「“パソコン”」を「“135”」，「“TV X1”」を「“248”」と書きかえて，プログラムを実行してみてください。数と数との足し算ではなく，文字列の結合が行なわれることがわかります。

●変数の名前

変数の名前は，その内容がすぐにわかるようにつけておくのがよいでしょう。ただし，従わなければならないルールがいくつかあります。

まず，変数の名前はどれもアルファベットではじまらなければなりません。最初の文字がアルファベットなら，後は英文字・数字・カタカナのどれでも使えます。グラフィックスなどの

記号は使えません。

最初の文字がアルファベットであっても使えない英数名もあります。それは，コマンドやステートメントなど，あらかじめＸ１が持っているBASICの言葉です。これらを予約語といいます。他にも，たとえば合計を「GOUKEI」や，「TOTAL」なども変数に使うことはできません。「GO」や「TO」の予約語が先頭についているからです。

これらの約束さえ守れば，240文字までの範囲で自由に変数の名前は決めることができます。しかし，あまり長い名前をつけてもメモリの無駄使いになりますから，わかる範囲で省略しておくことも必要です。

また，数値変数と文字変数には，「$」があるかないかの違いしかありません。ですから，「KOTAE」と「KOTAE $」は別のものとして扱われますが，あまりに長い変数名ではかえってわかりにくくなってしまいます。

●これが最初のプログラム!!

寄り道が少し長くなってしまいました。もう一度，チャンネルコントロールのためのプログラムを見てみましょう。

```
10 INPUT CH
20 CRT 0
30 CHANNEL CH
40 GOTO 10
```

10行でCHという変数に希望するチャンネルの数値を入れています。そのための命令がINPUT文です。変数CHにデータを入れなさいという意味になります。ですからプログラムを実行すると，Ｘ１は「？」マークを表示して「データは何ですか？」と問いかけてくるのです。

20行では，ディスプレイの画面をテレビに切りかえています。CRT文を，これまではダイレクトモードで使ってきましたが，このようにプログラムの中で使うこともできます。CRT 0は画面にテレビ放送を表示させる命令です。

30行では，CHANNELという命令でテレビのチャンネルを切りかえます。これもダイレクトモードを使ってきた命令ですが，プログラムの中で使っています。注目したいのは変数CHをCHANNEL文の後においていることです。キーボードから入力した数値をいったん変数CHの中にしまっておき，ここで改めてCHANNEL文の指示のために変数の中の数値を使うのです。

40行ですが，ここではGOTO文を使っています。この40行では10行へ行けと指示しています。GOTO文は次のような書き方をします。

GOTO 行番号

GOTO文の後につけた行番号へ進めという指示です。コンピュータは行番号の順に仕事を行なっていきます(というよりも，行番号はコンピュータにさせる仕事の順番なのですが)。GOTO文を使うことによって行番号での次の行にではなく，他の行へとプログラムを進めることができます。ここでは，プログラムの流れを10行に戻しています。Ｘ１は，10行で再びチャンネルの入力を待つことになります。40行のGOTOの文のおかげで，このプログラムでは何度でもチャンネルを切りかえることができるのです。

これでプログラムというもの，そしてそのしくみがだいたいわかりました。

●LISTでプログラムを確認

さて，もう一度今のプログラムを見てみましょう。その前にプログラムの実行を停止してディスプレイをテレビ放送から，コンピュータに切りかえなければなりません。プログラムの実行を停止するには，SHIFT キーと BREAK キーとを一緒に押します。ディスプレイテレビの画面の切りかえは，もうご存じですね。

では，LISTとキーインしてリターンキーを押してください。LISTは，プログラムを表示しなさいという命令です。しばしば使う命令なので，ファンクションキーの４番（F4）に，この命令が定義されています。プログラムを表示させることを「LISTをとる」といいます。このプログラムに次のような行を１つつけ加えます。プログラムの一番下のはずれにカーソルが点滅していてちょっと気になりますが，そこにいきなり書いてもかまいません。

5 PRINT "TV O MIYOU"

入力したら，もう一度LISTをとってください。行番号５はちゃんと行番号10の上にきています。Ｘ１は，行番号の順にプログラムの整理もやってくれるのです。

ここで，もう一度プログラムを実行してみてください。画面には，「TV O MIYOU」と表示され，その下に「？」マークが表示されます。PRINTは，画面に表示せよという命令です。画面に表示させる内容を「"」で囲って示してやります。また，

```
10 A=17
20 B$="サイ"
30 PRINT A,B$
```

のように変数の中身を表示させることもできます。どのようなプログラムでも必ずといってよいほど使われる命令なので，Ｘ１では「？」マーク１つで代用させることができるようになっています。SHIFT キーと BREAK キーを一緒に押して，もう一度プログラムの実行を止め，今度は次のようにキーインしてみます。

```
10 INPUT "CHANNEL";CH
```

このようにキーインしてリターンキーを押すと，これまでの10行が上のように新しく書き直されます。本当に書き直されたかどうか，LISTをとって確かめましょう。

F4 キーを押してプログラムを画面に表示させてみましょう。プログラムの10行が確かに書き直されていますね。

```
5 PRINT "TV O MIYOU"
10 INPUT"CHANNEL";CH
20 CRT 0
30 CHANNEL CH
40 GOTO 10
```

●プロンプト文でＸ１と対話

それではこのプログラムを実行しましょう。

「TV O MIYOU」と表示された下に，今度は「CHANNEL?」と表示されました。INPUT文までプログラムの流れがやってくると，Ｘ１はクウェスチョンマークを表示してデータの入力を求めます。しかし，クウェスチョンマークだけではプログラムを作った人はともかくとして，他の人には数値なのか，文字なのか，どのようなものを入力してやればよいのか分かりません。

そこで「CHANNEL」と画面に表示させておき，ここにはチャンネルを入力するのだと教えるようにしているのです。このような変数の注意書きを**プロンプト文**と呼びます。「”」で囲んだメッセージがそのまま表示されますから，プロンプト文をうまく使うとプログラムを実行したときの操作はとても楽になります。また，プロンプト文と変数の区切りに使っている「；」（セミコロン）を「，」（カンマ）にすると，クウェスチョンマークが表示されなくなりますからうまく使いわけましょう。

PRINT文とINPUT文は，これからのプログラムの中でもしばしば使います。そのつど使い方を理解するよう努めてください。

らんだむめも

変数名について

BASICでプログラムを作るときには，必ずといってよいほど文字列を入れる箱を用意しなければなりません。この箱を変数と呼びます。

プログラムが大きくなって変数がたくさんになると，整理がたいへんです。そこで，変数に名前をつけて，あなたがどの変数に何を入れたかわかりやすくします。この名前を変数名といいます。

変数名をつけるとき，注意しなければならないことがいくつかあります。

まず，変数名はアルファベットではじまらなければなりません。あとはアルファベットと数字（カタカナやグラフィック記号も使えます）を組み合せて240文字までの名前をつけることができます。EIGOとかS58ネンというように，わかりやすい名前をつけることもできるのです。

アルファベットではじめるということの他に，もう１つ大きな約束があります。X１のBASICには予約語といって，変数名に使ってはいけない言葉があります。予約語というのは，X１のBASICに登録されているRUN，GOTO，PRINTなどの命令（コマンド，ステートメント）および，CHR$，SIN，COSなどの関数のことです。また，TO，STEP，TABなども予約語に含まれます。

この予約語は変数名に使うことはできませんが，HGOTO 4，X１CHR$のように変数名の一部であればかまいません。ただし，このとき予約語を先頭にもってきてはいけません。GOKEIとかTOTALなどは，つい間違えてつけてしまいそうな例です。

このような約束を守れば，変数名は自由につけられますから，あなたのプログラムにあわせてわかりやすい名前にしましょう。ただ，あまり長いものは，メモリを多く使うことになりますし，打ち込むときにたいへんですので，できるだけ避けましょう。

3-2 IF文でアプローチ

●内蔵時計の時刻TIME$

いよいよ「テレビ24時間予約プログラム」の制作開始です。テレビ24時間予約プログラムはＸ１の機能を活かしてBASICの命令でテレビをコントロールしようというものです。

まず，希望の時刻にテレビのスイッチをONにすることを考えて，その部分だけをひきうける小さなプログラムを作ってみましょう。

Ｘ１には，本体やディスプレイの電源を切っても働きつづける内蔵時計があります。この時計が希望の時刻を示したときにTVPW ONの命令を実行させればよいですね。

```
10 PRINT TIME$
```

と入力してみましょう。行番号がついていますから，これはプログラムの一部分です。

PRINT文は，画面に文字や記号，変数の値や計算の結果などを表示しなさいという命令文です。

TIME$は，Ｘ１の内蔵時計の時刻を呼び出す関数です。TIME$は変数の一種で内蔵時計の時刻を入れる専門の箱だと思ってください。このプログラムはTIME$という箱の中の値(つまり，現在の時刻)を表示せよ，という意味になります。それではRUNさせてみましょう。画面には現在の時刻が表示されます。

たとえば，15時14分12秒と表示されたとしましょう。それでは15時20分にテレビのスイッチを入れるにはどのような方法があるでしょうか。TIME$の中身が15：20：00になったときに，「TVPW ON」の命令を実行させればよいでしょう。Ｘ１に現在の時刻が15：20：00であるかどうかを判断させるという作業がこのプログラムの中心になります。

●コンピュータは考える

コンピュータの持つもっとも優れた能力の１つが，この「判断」や「選択」の機能です。私たち人間が，「晴れていたら，野球をしよう」「バーゲンセールで安いから，靴を買おう」と考えて行動するのと同じように，定められた条件に従って判断することがコンピュータにはできるのです。

どのような方法で判断や選択を行なうのかを見ていきましょう。

ある条件を判断させようとするときBASICではIF文というものを使います。IFとは，英語の「もしも」です。IF文は，

IF～THEN…

というように，常にTHENと一緒に使います。THENというのは，やはり英語で「それでは」「そのとき」という意味です。～には条件を

表３－１　比較演算子

記号	書き方	意味
＝	A ＝ B	AとBは等しい
＞	A ＞ B	Aの方がBより大きい (Bの方がAより小さい)
＜	A ＜ B	Bの方がAより大きい (Aの方がBより小さい)
≧	A＞＝B A＝＞B	Aの方がBより大きいか， または等しい
≦	A＜＝B A＝＞B	Bの方がAより大きいか， または等しい
≠	A＜＞B	AとBは等しくない

※この「＝」は，代入文ではない．従って変数Ａに変数Ｂを代入するという意味ではない．

書き，…には条件が成りたったときに行なう仕事を書きます。

コンピュータはコンピュータの言葉で考えますから，条件も２つの変数の比較をするという独特の書き方をしてやらねばなりません。たとえば「IF A＝＞B」(もしも，AがB以上だったら)，「IF X＜10」(もしもXが10未満だったら)という形です。この比較の方法を表にまとめてみましょう（表３-１)。

条件が成り立たないときには，THENから後の「…」を無視して次の行まで進みます。BASICのプログラムは行番号の順に実行されますから，次の行に別の仕事を書いておけば，IF文の条件が成り立たなかったときにはＸ１はそちらの仕事をするわけです。明日晴れたら野球をしよう，晴れなかったら映画を観ようというのと同じです。さっそく試してみましょう。

```
20 IF TIME$="15:20:00" THEN TVPW ON
```

RUNさせてみましたか？　よほど運がよい場合を除いて，テレビのスイッチがONになることはありません。これは，IFの後の条件が成り立たなかった場合のことを考えていないためです。晴れたら野球をしようと決めていたが，雨が降ったときのことを考えておかなかったというようなものです(RUNさせた時間にたまたま条件が成り立てば，テレビのスイッチが入りますけれど)。それでは，次のようにキーインしてください。

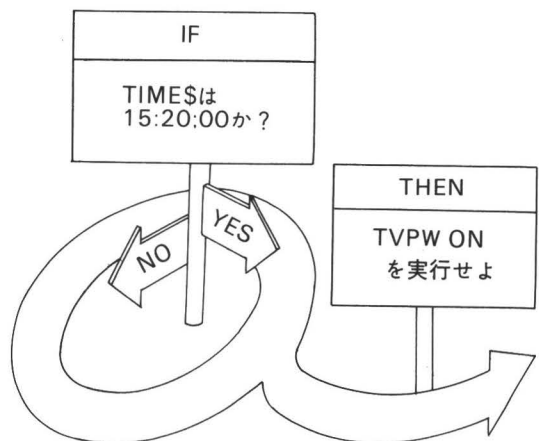
```
30 GOTO 10
```

これが10行へ行けという指示であることは，すでに述べました。実行するまえにプログラムの全体を見てみましょう。

```
10 PRINT TIME$
20 IF TIME$="15:20:00" THEN TVPW ON
30 GOTO 10
```

10行で，現在の時刻を画面に表示し，20行でその時刻が15時20分00秒であるかどうかの条件を判断しています。ここで「"」を使う理由は後にまわして，IF文の勉強を先にすませてしまいましょう。条件が成り立たないときには，THENから後を無視して30行へ進み，30行では再び最初の10行へ戻ります。このプログラムは，20行の条件が成り立つまで同じ１つの環の中をぐるぐると回りつづけています。このようなプログラムの環を「ループ」といいます。そして，IF文の条件が成り立つとTHENの後ろにある「TVPW ON」を実行するというわけです。

ところで，TVPW ONを実行した後，このプログラムはどうなったのでしょうか。

プログラムは必ず行番号の順に実行されるという大原則がありました。20行のIF文で条件が成立して，テレビのスイッチをONにした後，このプログラムは原則どおり20行へ進んだのです。そして30行から10行に戻り，もう一度IF文の条件を判断して30行へ進みまた30行から10行に戻り……これでは終わりがなく，プログラムは同じ環の中を無限に回りつづけてしまいます。いつまで待っても，疲れ知らずのコンピュータですから仕事を途中でやめてしまうということがありません。このように終わりがないプログラムの環を「無

限ループ」といいます。

無限ループに入ってしまったプログラムの実行を止めるには，SHIFT キーと BREAK キーを一緒に押すか，CTRL キーと C キーを一緒に押します。

●ENDとGOTO

IF文を使うときには，

①条件が成立したとき

②条件が成立しなかったとき

③条件が成立し，決められた仕事を終わった後の処理

の3つの点に注意をして，どの場合にも不都合がないようにしておかなければなりません。ここでは③を忘れていたわけです。テレビのスイッチをONにすればもうこのプログラムを動かし続ける必要はありません。プログラムの実行を終わりなさいという指示も，BASICの中に用意されています。END文です。END文は文字どおり終わりを示し，ここまできたらどんなプログラムも止まってしまいます。次の2つのプログラムを前のプログラムとよく比べてみてください。

プログラムⒶ

```
10 PRINT TIME$
20 IF TIME$="15:20:00" THEN TVPW ON:END
30 GOTO 10
```

プログラムⒷ

```
10 PRINT TIME$
20 IF TIME$="15:20:00" THEN GOTO 40
30 GOTO 10
40 TVPW ON
50 END
```

プログラムⒷでは，GOTO文の使い方に注意してください。IF文の条件が成り立つまで30行から10行へ戻るループの中をプログラムはぐるぐる回りつづけます。条件が成り立つと，40行へ飛んでテレビの電源をONにして50行で終わります。

らんだむめも

カーソル移動の魔法の杖

EDIT文などを使って行の中身を修正する場合，その命令の先頭までカーソルを移動させなければなりませんが，このようなときに CTRL ＋ F キーを押すと，カーソルは1文字ずつではなく，1つの命令文ずつ進んでいきます。進み方は，まず行番号，1番目の命令文の先頭，次にその命令文のパラメータという順に進みます。ただし，スペースを省略して詰めて書かれている場合，このようになるとは限りません。行きすぎたり，前に戻したいときは，CTRL ＋ B キーを押すと，この反対に動きます。

もう1つ，すでにある行と行の間に新しい行を追加したい場合，普通はカーソルを画面の下の方に移動させていき，そこで入力します。この方法では，うっかりすると行番号を同じにしてしまって，前のものを消してしまったりすることもあります。このようなとき，CTRL ＋ O キー，または CTRL ＋ N キーを使うと，表示されている行と行の間をあけることができます。CTRL＋Oは，カーソルのある行から下を下げる働きを，CTRL＋Nは，この逆にカーソルのある行から上を上げる働きをします。

プログラムⒶの20行の中には，TVPW ONとENDという２つの命令が並んでいます。ENDは，文字どおりプログラムの終わりを示す命令です。BASICでは「：」(コロン)を使うことによって，いくつかの命令を１行にまとめて一緒に書くことができるのです。このような書き方を，**マルチステートメント**と呼びます。メモリを節約したいときなど，とても役立つ書き方です。またこの場合のようにIF文の中でいくつかの命令をいっぺんに実行させたいとき必ず使われる書き方です。

この「：」(コロン)や「，」(カンマ)，「；」(セミコロン)を**セパレータ**といいます。セパレータとは「分けるもの」という意味です。

3-3 時刻を決める、チャンネルを決める

●文字変数で時刻を決める

前のプログラムでは，テレビのスイッチをONにしたい時刻を，プログラムの中に書き込みました。何回も繰り返してテレビのスイッチをONにするためには，いく度もプログラムを書き直さねばなりませんから，これでは実用的なプログラムとはいえません。ここではプログラムを実行するたびに希望の時刻を自由に決められるようにしましょう。

前にテレビのチャンネルを選択するプログラムでINPUT文を使いました。INPUT文はプログラムを実行すると，クウェスチョンマークを表示してデータの入力をうながすステートメントです。これを使ってテレビを見たい時刻を入力するのが良さそうです。

INPUT文の中に希望する時刻を入れてやれば良いでしょう。ここではテレビのスイッチをONにしたいと希望する時刻をTC$という変数に入れることにします。

プログラムは次のようになります。

```
10 INPUT TC$
20 IF TIME$=TC$ THEN TVPW ON:END
30 GOTO 20
```

このプログラムをRUNさせると10行のINPUT文によって，最初にクウェスチョンマークが表示されます。時刻は内蔵時計の表示する時刻と同じ書き方でキーインします。たとえば「15：50：00」というようにキーインします。これは数字の集まりですが，電話番号などと同じく数字の文字列です。ですから変数もTC$というように，「$」マークをつけて文字列変数としています。リターンキーを押すと，TC$という変数の中にこの文字列が入ります。このプログラムではTIME$とTC$の比較を繰り返し続けて，2つが一致するとテレビ放送に画面を切りかえています。

非常に簡単なプログラムです。今度はテレビのチャンネルも自由に決められるようにしましょう。前にCHANNEL文を使ったプログラムを作りましたから，それを利用します。CHANNEL文は，テレビのチャンネルをCHという変数に入れておき，CHANNEL命令を使って「CHチャンネルに切りかえろ」という指示をプログラム中で行なえばよいのです。変数が2つに増えますから，プロンプト文を使って，プログラムをわかりやすいものにしておきましょう。

カナを入力するときにはスペースバーの横にある[カナ]キーを押します。もう一度[カナ]キーを押すと，アルファベットや記号が使えるようになります。

```
10 INPUT"ﾅﾝｼﾞ ﾆｼﾏｽｶ";TC$
15 INPUT"ﾁｬﾝﾈﾙﾊ";CH
20 IF TIME$=TC$ THEN TVPW ON:CHANNEL CH:END
30 GOTO 20
```

入力し終わったら，さっそくプログラムを実行してみてください。

●コードナンバー1550106 ～配列～

希望する時刻にテレビをONにし，希望するチャンネルに合わせることができるようになりました。今度は希望する時刻にテレビをOFFにしましょう。

まず，テレビのスイッチをONにしたい時刻を，前と同じようにTC$という変数に入れておきます。しかし，スイッチをOFFにする時刻は，別の変数に入れなければなりません。INPUT文とIF文を使って新しくプログラムを組みます。

```
10 INPUT "SW ON TIME (HH:MM:SS)";TVON$
20 INPUT "CHANNEL (1-12)";CH
30 INPUT "SW OFF TIME (HH:MM:SS)";TVOFF$
40 IF TIME$=TVON$ THEN TVPW ON:CHANNEL CH:GOTO 60
50 GOTO 40
60 IF TIME$=TVOFF$ THEN TVPW OFF:END
70 GOTO 60
```

INPUT文を使ってテレビのON/OFF，チャンネルの指定の３つのデータを入力し，後はIF文を使って，それぞれ指定された時刻と内蔵時計の示す時刻との比較をする基本どおりのプログラムです。それぞれのIF文で条件が成り立ったとき，成り立たなかったときの処理に注目してください。

40行のIF文で，変数TV ON$の値，つまりテレビをONにしたい時刻と内蔵時計の時刻とが一致すると，TVPW ONの命令を実行し，さらにCHANNEL CHを実行し，60行へ進めという命令を実行します。時刻が一致しなければ，50行のGOTO文でプログラムの流れは，60行での処理と同じでIF文へ戻ります。

これで，スイッチのON/OFFとチャンネルの切りかえができるようになりました。プロンプト文のおかげで操作もずいぶんとわかりやすくなっていることが，実際にプログラムを走らせてみると確かめられます。しかし，スイッチのON/OFFもチャンネルの切りかえも１回だけしか行なえません。これでは，わざわざコンピュータを使う必要はありませんね。24時間，予約したとおりにコンピュータでテレビをコントロールしたいものです。そのためにはどのようなプログラムを作ればよいのでしょうか。

「つぎつぎに変数を作って，その変数の１つ１つをIF文でチェックしていけばよさそうだ」ということはわかります。しかし，こう考えた場合，そのいくつもの変数名をどのように用意するかが問題になります。

変数名の問題を解決する前に，プログラムの中でコントロールしなければならないのは何なのかをもう一度確かめておきましょう。

①時刻

②テレビのON，またはOFF

③テレビのチャンネル

この３つの要素をばらばらにではなく，うまく１つにまとめることができるとプログラム作りは楽になりそうです。

●コードナンバーってどんなもの

近ごろではいろいろなところでコードナンバーが使われています。**コードナンバー**というのはいくつかの要素を数字で表現したものです。みなさんの中にも銀行のキャッシュカードを持っている人がいることでしょう。キャッシュカードにはいくつかの数字が刻まれて並んでいます。これもコードナンバーの一種です。図書館では１冊ずつ全部の本に，異なるコードナンバ

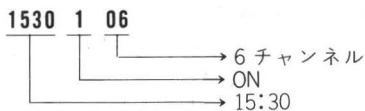
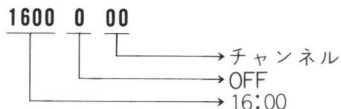
ーをつけて整理分類をしています。歴史，社会，自然科学，芸術などの区別をした後，さらに時代や国や分野によって細かな区別をするのですが，そのすべてが0から9までの数字の組み合わせで表わされています。たとえば，「2151」というナンバーはそれぞれの数字が「2（歴史），1（日本），5（中部地方），1（山梨県）」という意味を持っていて，これは2151冊目の本という意味ではありません。コードナンバーをうまく利用すれば，何万冊，何10万冊の本でもしっかりと分類し，きれいに整理することができるのです。

それでは，コードナンバーを使ってこのプログラムに必要な3つの要素を1つにまとめてみましょう。

まず最初に，4桁の数字で時分を表わすことにしましょう。このとき，時や分が1桁の場合，その前に0をつけるようにします。文字列の文字数を表わす関数としてLENがあります。これを利用して桁数を調べ，もし1桁ならば前に“0”を加えるようにします。

```
10 INPUT "ナン時",JI$
20 IF LEN(JI$)=1 THEN JI$="0"+JI$
```

スイッチのON/OFFは，ONを1，OFFを0で表わすことにします。チャンネルは1から12までありますので2桁分用意し，1桁の場合は同様に前に0をつけることにします。

こうして，たとえば「15時30分にテレビをONにして6チャンネルを見る」という場合のコードは，

となります。30分テレビを見て16時にテレビをOFFにする時のコードは，

です。テレビをOFFにするときの，チャンネルのコードは何でもよいわけですから，00としておきました。

これで，1つの文字列の中にテレビをコントロールするための3つの要素をまとめることができました。

●文字をつないでコードナンバー

```
10 INPUT "SW ON TIME (HH:MM:SS)";TVON$
20 INPUT "CHANNEL (1-12)";CH
30 INPUT "SW OFF TIME (HH:MM:SS)";TVOFF$
40 IF TIME$=TVON$ THEN TVPW ON:CHANNEL CH:GOTO 60
50 GOTO 40
60 IF TIME$=TVOFF$ THEN TVPW OFF:END
70 GOTO 60
```

これがこれまでに作ったプログラムの全体です。ここから必要な要素をぬき出してコードナンバー化しなければなりません。

文字列同士を「＋」記号で結合することができる，ということをみなさんは覚えていますか。

時刻＋1（ON）／0（OFF）＋チャンネル

とすれば，コードナンバーを作ることができそうです。ここで注意しておかなければならないことが2つあります。1つは，チャンネルの指定を数字の文字列として処理するということです。文字列同士の結合ですからチャンネルの数字を数値変数にではなく，文字列変数に入れるようにします。もう1つはテレビのON/OFFを数字の1と0で指定するということですが，これも数字の文字列として扱わなければなりません。また，時刻の指定も「時」＋「分」というようにしておいた方が，後の処理が楽になります。間に「：」が入るとコードとしても不自然です。

コードナンバー化の問題はコードの分解とあわせて考えた方がよくわかると思いますので，まずここではコード化の原則だけを覚えておきましょう。

●配列変数でたくさんの箱

ここで変数の問題に戻りましょう。この文字列を収めておくための変数を用意しなければなりません。

テレビのON/OFFをしたり，チャンネルを切りかえるとなるといくつもの変数を用意しなければなりません。そこで**配列変数**というものを使うことにします。

前に変数というのは，データを入れておく箱であると説明しました。今までに使った変数はすべて，箱が1つだけのものです。配列変数ではたくさんの箱を，まとめて1つの名前で用意してしまうことができます。1つの名前で用意されたたくさんの箱の1個1個に番号をふったものと考えられます。

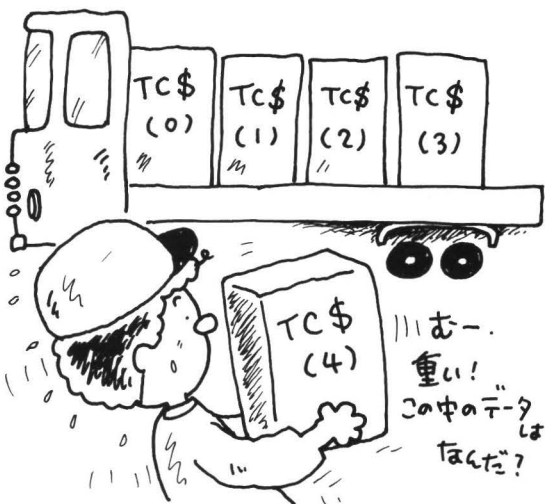
配列変数の書き方の例を1つあげます。

TC$（4）

TC$というのがこの配列変数の変数名です。「$」がついていることで，これが文字変数の仲間であることがわかります。これまで勉強してきた変数と同じく，配列変数にも数値を入れるためのものと文字列を入れるためのものがあります。ここで扱うコードナンバーは文字列をつないだものですから，変数も文字列変数でなければなりません。

さて，変数名の後ろに，数字の入ったカッコが書かれています。カッコの中に書かれた数に従って，変数の箱が用意されるのです。

このカッコの中に書かれた数を**添字**（そえじ）といいます。この添字の数だけ箱が用意されます。ただ，ここで注意してほしいのは0も1つの箱として用意されるということです。たとえばこの例でいえば，TC$という1つの変数名でTC$（0）からTC$（4）までの5つの箱が用意されることになります。そして，この5個の箱がそれぞれに別々のデータを収めておくことができます。

TC$(255)とすれば，256個の箱が用意されることになります。実際にプログラムを作るときに

は，むやみにたくさんの箱を用意すればそれでよいというわけではありません。いくつの箱を用意するかを，最初に決めておかなければなりません。

●DIM文で配列宣言

用意する箱の数を決めるには，DIM文を使います。DIM文の書き方は，次の例のようになります。

```
DIM TC$(4)
```

DIM文が配列変数で用意する箱の数を指定します。ですから，数値変数とDIM文とはいつでも1つの組になって使われます。ただし，用意しなければならない箱の数が10までのときにはDIM文を使わなくても配列を使うことができます。

この配列宣言の前にOPTIONBASE 1と宣言しますと，配列の添字の一番小さい値(下限)を1にすることができます。通常添字の下限は0になっています。配列を宣言する前に一度だけ宣言でき，再宣言することはできません。

配列変数で用意される箱のことを**配列要素**といいます。また，箱の数を決めることを**配列宣言をする**といいます。DIMとはDimensionの略で宣言の意味なのです。

DIM文は，その配列を使う前に一度だけ使用し，後から箱の数を変えたりすることはできません。

さて，この例ではTC$という変数名で配列の番号0から4まで，5つの箱が用意されることになりました。箱は用意されましたが，中身はまだ空っぽです。この空っぽの配列変数の箱の中に順番にデータを入れていかなければなりません。どのような方法があるでしょうか？

3-4 まわりまわって、またまたまわる

1日に30回のテレビのON/OFFをするとして，30個のコードを入れる箱を用意しなければなりません。配列変数の箱を30個用意しましょう。変数名をこれまでどおりTC$とすると，配列宣言はこのようになります。

DIM TC$(30)

配列変数の中に順番にデータを入れていく方法ですが，便利な命令がありますので紹介しましょう。

```
10 DIM Tc$(30)
20 FOR I=1 TO 30
30 INPUT Tc$(I)
40 NEXT I
```

この小さなプログラムで配列変数TC$のTC$(1)からTC$(30)までの30個の箱に順番にデータを入れていくことができます。便利な命令というのは，ここで使っているFOR～TO～NEXTという命令です。

ここでは，Iの値を1から30まで1ずつ順番に増やしながら，そのつど配列変数TC$の添字に値をあてはめていって，30個の箱の中に値を入れていきます。

配列変数TC$の中のデータを読み出すように，次のプログラムをつけ加えてください。

```
50 FOR I=1 TO 30
60 PRINT Tc$(I)
70 NEXT I
80 END
```

プログラムを実行するとクウェスチョンマークが30個表示されますから，確かにTC$という変数に30個のデータが収められているらしいことがわかります。50行から後のプログラムが実行されると，そのデータが入力した順番に表示されます。

30回も入力するのでは疲れてしまいますか？　それでは希望する数のデータだけを入れられるようにしましょう。

```
10 DIM Tc$(30)
15 INPUT E
20 FOR I=1 TO E
30 INPUT Tc$(I)
40 NEXT I
50 FOR I=1 TO E
60 PRINT Tc$(I)
70 NEXT I
80 END
```

まず配列変数ＴC$を30個用意します。１日にテレビのON/OFFをする回数は最大30回となります。

プログラムのどこをどう変えているのか注意してみておいてください。

TV24時間予約プログラムの考え方もこのプログラム例と同じです。時刻・チャンネル・ON/OFFの３つの要素をまとめたコードを配列変数の中に入れておき，再び順番に読み出せばよいのです。

FOR〜TO〜NEXTの命令は一般に，

FOR V=A TO B STEP C〜NEXT

という形で使います。この例では，「変数(V)の値をＡからＢまで順番にＣずつ増やしなさい」という意味になります。Ｃの値を負（マイナス）にしてやれば，順番に数を減らしていくこともできるのです。ただし，この場合はＡの値の方がＢの値よりも大きくなければならないのはもちろんです。また，Ａ，Ｂ，Ｃの値を変数で指定することもできます。STEPを省略するとＣの値は自動的に１となります。

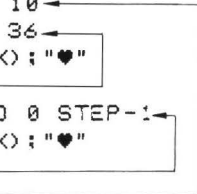
変数Ｖの値は，最初はＡですが，プログラムの流れがNEXTの命令までたどりつくとFOR〜TOのところまで戻ってＣだけ増えます。つまり，FOR〜TOからNEXTまでがループになっているのです。前に「無限ループ」という言葉が出てきました。FOR〜TO〜NEXTのループは，回数を限られてぐるぐると回るループです。

FOR〜TO〜NEXT命令はこれからもよく使いますので，どのような働きをするかいろいろと試しておきましょう。

```
10 FOR I=32 TO 255
20 PRINT CHR$(I);
30 NEXT I
```

これは，Ｘ１で使える文字や記号をすべて画面に書き出させるためのプログラムです。

20行にあるCHR$(１)というのは配列変数ではなく，関数というものです。この関数については《3-6》のところでくわしく説明します。パソコンでは文字や記号のそれぞれに番号がつけられており，いわば文字や記号の背番号といえます。この番号を**アスキーコード**といって，０から255までの通し番号が割りあてられています(０から31までは，文字や記号ではなく**制御コード**というものに割りあてられています。巻末アスキーコード表参照)。CHR$は，文字の背番号アスキーコードから文字や記号を読み出す関数です。

このプログラムでは，アスキーコードをＩという変数に入れておいてFOR〜TO〜NEXT命令で32から255まで変化させながら，CHR$(32)からCHR$(255)まで順番に文字や記号を画面に表示させます。

```
5 S=0
10 PRINT"AカラBマデ ノ タシザン"
20 INPUT"A=";A
30 INPUT"B=";B
40 FOR I=A TO B
50 S=S+I
60 NEXT I
70 PRINTA;"AカラBマデ ノ ゴウケイS=";S
```

このプログラムは，キーボードから入力したAからBまでの数の合計を求めるプログラムです。FOR～TO～NEXT命令の使い方をみると，Aの値がBの値よりも小さくないとうまく計算されないということがわかります。ループの中にはS＝S＋Iという代入式がありますが，ここではどのような処理をしているのでしょう。

変数Sの値は最初は0ですが，FOR～TO～NEXTのループを1回くぐりぬけたときには，Iの値があらたに代入されます。Iの値は最初はAとなっています。Iの値が代入されている変数Sが2回目にループをくぐりぬけたときには，さらにSの値に次のIの値（1増えています）が加えられます。このようにして，Iの値がBになるまで足し算を繰り返します。

次のプログラムは，題して「揺れるハート」です。

```
10 FOR K=1 TO 10
20 FOR X=0 TO 36
30 PRINT TAB(X);"♥"
40 NEXT X
50 FOR X=36 TO 0 STEP-1
60 PRINT TAB(X);"♥"
70 NEXT x
80 NEXT K
```

この例では変数KのFOR～TO～NEXTのループの中に，もう1つの変数XのFOR～TO～NEXTのループが入っています。このような形をFOR～TO～NEXTの**ネスティング**といいます。ネスティングでは2つのループが交錯してはいけません。内側のループが，必ず外側のループの中に収まっていないとエラーになってしまいます。

さて，FOR～TO～NEXT命令のひととおりの働きがわかったところで，本題のテレビ24時間予約プログラムに戻って話を進めましょう。

このプログラムでは配列変数の中に，テレビをコントロールするためのコードを入れることにしました。その方法もわかりましたし，コードを読み出す方法もわかりました。しかし，テレビをコントロールするには，コードを読み出すだけではだめです。読み出したコードを①時，分，②チャンネル，③ON/OFFの3つの要素に戻してやって，それに対応する命令を実行させなければなりません。次々と難問がわいてきますが，一歩一歩着実に進むことにしましょう。

らんだむめも

行番号を書く手間をはぶこう

キーボードからプログラムを打ち込んでいくときに，いちいち行番号をつけるのは面倒です。このようなとき，AUTO文を使うと便利です。行番号の作り方は，AUTO　最初の行番号，行番号をいくつずつ増やすかの指定をします。たとえば，

AUTO　100，5

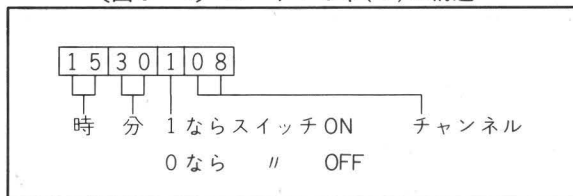
とすれば，最初の行番号が100になります。その後はリターンキーを押すたびに，行番号は5ずつ増えていきます。増やしていく数の指定を省略すると，最初の行番号が10，増やしていく分は10となります。

すでにプログラムが入力されているときにAUTO文を実行すると，入力済みの行が表示されて，その行の先頭にカーソルが表われますから，何行も続けて修正するときは便利に使うことができます。[SHIFT]＋[BREAK]を押すと，AUTO文は解除されます。

3-5 コードの分解調査

ここでは，この一連のコードを作ったときに元の文字列に分解する方法を考えます。

まずコードナンバーの構造をもう一度確認しておきましょう（図3－1）。

このコードからテレビをコントロールする命令のそれぞれを調べるためには，文字列の一部分を取り出す作業が必要です。たとえば，時刻のうち時の単位を知るにはコードの左から2文字を取り出さなければなりません。チャンネルは右側の2文字になります。BASICにはこのような処理をするために文字列処理関数（文字関数）というものが用意されています。例をあげると，A$＝“コンニチハ”という文字列の左から4文字を取り出すには，

〔図3－1〕 コードTC$(I)の構造

B$＝LEFT$(A$, 4)（レフトダラー）

としてやると，B$にA$の左側4文字，つまり“コンニチ”が入ります。

```
A$="コンニチハ"
OK
B$=LEFT$(A$,4)
OK
PRINT B$
コンニチ
OK
```

カッコの中のカンマの後ろで取り出す文字数を指定しています。

右側の何文字かを取り出すには，RIGHT$（ライトダラー）(A$, n)とすればよいだろうと想像がつきます。試してみましょう。

```
A$="コンニチハ"
OK
C$=RIGHT$(A$,3)
OK
PRINT C$
ニチハ
OK
```

もし，指定する文字数nが，元の文字列A$の文字数より多い場合，LEFT$，RIGHT$とも元の文字列全部になります。

もう一度テレビコントロールのコードを見てください。スイッチのON/OFFや分の単位は，左右からではなく途中の何文字かを取り出さなければなりません。このような場合には，MID$（ミドルダラー）を使います。たとえば，A$の左からn文字目からのm文字分を取り出すためには

B$＝MID$(A$,n,m)

とします。

```
A$="コンニチハ"
OK
D$=MID$(A$,3,2)
OK
PRINT D$
ニチ
OK
```

mを省略すると，n文字目から右側のすべての文字列を取り出します。これらの文字関数を使ってコードを時，分，スイッチの入切，チャンネルのそれぞれの要素に分解し取り出すことができます。

```
810 JI$=LEFT$(Tc$(I),2)
820 FU$=MID$(Tc$(I),3,2)
830 SW$=MID$(Tc$(I),5,1)
840 CHAN$=RIGHT$(Tc$(I),2)
```

これで，4つの変数にコードを分解することができました。ここまでくればしめたものです。後は4つの変数を使って実際にテレビをコントロールする部分を考えればよいのですから。

ところで，テレビをコントロールするのに必要な要素は時刻・チャンネル・ON/OFFの3つでした。ここでわざわざ4つの変数にコードを分解しているのはなぜでしょう。実は時刻を時と分との別々の変数に分けた方が，プログラムの処理も操作も簡単になるからなのです。2つに分けないとするとプログラムがどのように変わってくるかは，みなさんで考えてみましょう。

変数からテレビをコントロールする方法は，このプログラムを作りはじめるところでみておきました。しかし，あのときにはたった1つの変数を比較するだけでしたが，今度は4つの変数の比較をしなければなりません。

TIME$の時の単位を表わす部分と，JI$という変数とをIF文で比較します。これら2つの値が同じになっていたら次は，FU$と分の単位を比較するという手順になります。これを逆に考えてみると，JI$とTIME$の値が異なっていれば，それから後の分の比較やスイッチのON/OFF，チャンネルの切りかえといった判断は必要ないことがわかります。また，時が一致しても分の単位で違っていればやはりその後の判断は必要なくなります。このような場合，不要な判断の部分は飛ばしてしまって差しつかえありませんから，次のコードを調べるようにします。

TIME$から時，分の部分を取り出すにはLEFT$，MID$を使いますが，TIME$では「15：30」のように時，分の間に「：」（コロン）がありますから注意してください。

次にスイッチのON/OFFですが，ONを1，OFFを0と決めてありました。0でない場合は必ず1になっているというわけです。このような場合，IF文と組み合わせて，ELSEを使うと便利です。

```
870 IF SW$="0" THEN TVPWOFF:GOTO 900 ELSE TVPWON
```

IF～THEN～ELSEでは，条件が成り立ったときのTHENの後の処理と，成り立たなかったときに行なうELSEの後の処理という，2つの処理を一度に指定することができます。ここでは，もし(IF)SW$が0なら(THEN)スイッチを切れ，　と命令しています。SW$が1ならELSEの後の命令に従ってスイッチを入れ，それがすむと次の行の仕事に移ります。

スイッチを入れたら，次の仕事はチャンネルの切りかえです。CHANNEL文を使うことはいうまでもありません。チャンネルはCHAN$という変数に入っています。ところが，試してみると

わかるように，CHANNELという命令の後に「＄」マークのついた文字変数を置くとエラーになってしまいます。

```
CHAN$="4"
OK
CHANNEL CHAN$
Type mismactch
OK
```

文字列変数CHAN$の中の数字文字列を，なんとかして数値にしてやらなければなりません。これは，VALという関数を使います。これは文字列を数値に変換する関数です。

```
880 CHAN=VAL(CHAN$)
890 CHANNEL CHAN
```

さあこれでどうやらコードからテレビのコントロールを行なう部分ができあがったようです。プログラムがどのように流れていくかをしっかり確かめてください。

```
800 FOR I=1 TO EF
810 JI$=LEFT$(Tc$(I),2)  ← コードから「時」の部分を取り出す。
820 FU$=MID$(Tc$(I),3,2) ←     〃     「分」     〃
830 SW$=MID$(Tc$(I),5,1) ← スイッチON,OFFを指示する。0と1を取り出す。
840 CHAN$=RIGHT$(Tc$(I),2) ← チャンネルの数字を取り出す。
850 IF JI$<>LEFT$(TIME$,2)  ← TIME$の「時」の単位とコードの「時」が違っていれば
GOTO 900                       後の仕事は必要ないので次のコードにいく。
860 IF FU$<>MID$(TIME$,4,2)
GOTO 900                    ← 分についても同様
870 IF SW$="0" THEN TVPWOFF ← 0ならスイッチをOFFにする。チャンネルはどうでもよ
:GOTO 900 ELSE TVPWON          いので次のコードへ、1ならスイッチをONにする。
880 CHAN=VAL(CHAN$)         ← 文字列になっているチャンネルの数字を数値に変換する。
890 CHANNEL CHAN            ← チャンネルを切り換える。
900 NEXT I
```

3-6 プログラムのパーツ作り

●データのチェック

ここまで，一度にいろいろなことが出てきて，だいぶむずかしかったかもしれません。ここでちょっと，「テレビ24時間予約プログラム」の整理をしてみましょう。

○配列を宣言する
○画面をきれいにする

データの入力部分
○INPUT文で入力
○時，分，スウィッチ　チャンネルをまとめてコード化
○コード化したものをFOR～NEXT文で配列変数に入れる

テレビのコントロール部分
○配列変数からFOR～NEXT文でコードを読み出す
○コードを分解して時,分,スイッチチャンネルに対応する変数に分ける
○それらの変数をIF文で判断しテレビのコントロールを実行する

こうしてみると，「テレビ24時間プログラム」の骨組がだいたいわかってきました。ここでは，この骨組に肉づけをしていって，実際にプログラムを実行させる用意をしましょう。

まず，入力されてくるデータをチェックできるように，プログラムに手を加えておきます。

テレビをコントロールするためのデータは限られた値で，たとえば，入力されるチャンネルは1から12の範囲にあるはずです。もし，それ以外の誤ったデータがそのまま受けつけられてしまうと，それまで順調に働いていたプログラムも，テレビをコントロールする段階になってエラーを出すことになってしまいます。これを避けるために，データを入力した時点で，それが誤ったデータならばもう一度入力し直させるようにすればいいわけです。

人間が入力や操作の誤りをすることはしばしばあることです。誤った入力データを受けつけないようにプログラムの方で工夫をしておくことは，その後，プログラムを実行していく際のトラブルを避ける意味で，実はとても大切なことです。

●関数は加工機械

次の例を見てください。

```
270 IF VAL(CHAN$)>12 OR VAL(CHAN$)<1 OR FRAC(VAL(CHAN$))<>0
THEN GOTO 260
```

IF文を使って，チャンネルが正しく入力されたかをチェックする部分です。VALという関数は前に説明したように，数字文字列を数値に変換するものです。FRACも関数で，これはその後の数値の小数部分を与えます。

関数というのは一種の命令ですが，コマンドやステートメントなどの命令とは異なります。関数はその後に，カッコで囲まれた数値や式，文字列などを別の数値や文字列に直す働きをします。つまり，文字列や数値，式の変換や加工をさせる命令というわけです。

みなさんもコピーの機械を使ったことがあると思いますが，白いコピー用紙に文字や絵が映し出されてきます。今日ではさまざまなコピー機があって拡大や縮小も行なえます。関数による変換前の数値や文字列がコピーの原稿で，機械の働きで拡大されたり縮小されたりの加工を

されてコピー用紙に映し出された絵や文字が，変換後の数値や文字列にあたります。

Ｘ１には，数値関数，文字関数，特殊関数，入力文字関数，タイマー関数，システム関数といったものがあらかじめ用意されています。これを組み込み関数といいます。このほかに，コンピュータを使う人が自分で関数を定義することができるようになっています。こちらは，ユーザー定義といいます。

話を元に戻しましょう。ここで「OR」という言葉を使っています。１より小さいか12より大きいか，また小数部分が０になっているか(つまり整数か)，という判断を合理的に処理しています。ORは英語で「または」，「あるいは」という意味ですから，たとえば，

```
IF A=1 OR B=1 THEN～
```

とすれば，変数Ａの値が１，または変数Ｂの値が１または両方が１ならば，THEN以下の仕事をする，ということになります。ORで結ばれた条件を論理和といいます。ORのほかにも，

```
IF A=1 AND B=1 THEN～
```

とすれば，ＡもＢも共に１のときに，という意味になり，このANDで結ばれた条件式を論理積といいます。このORやANDは，コンピュータの基礎となる考え方ですし，これからもよく使う手法ですのでよく覚えておいてください。

●画面の整理

画面に表示されたチャンネルや時刻などのデータが，正しいかどうかを入力した人が自分で確認できるようにしましょう。正しければ[Y]キーを押し，正しくなければ[N]キーを押すというようにすれば，入力を誤ったときにも訂正ができます。もちろん[Y]キーが押されたときに，はじめてデータが配列変数に代入し直されるようにしなければこの工夫も意味がなくなってしまいます。

押されたキーの判断は，INKEY$という特別な関数を使って行ないます。INKEY$はキーボードの見張り役です。INKEY$は，変数のようにキーボードから入力された１文字を，自分の箱の中にしまってしまいます。

```
10 IF INKEY$="Y" THEN PRINT "Yes" ELSE PRINT "No":GOTO 10
```

とすると，キーボードから“Y”が入力されるまで「No」と表示し続けます。キーが押されると，つまり文字や記号が入力されると，それをINKEY$関数の中にいったんしまってからIF文で文字列“Y”と比較します。比較した後は，THEN，ELSEの後の命令を実行します。GOTO文で同じ行の先頭に戻るというのがおもしろいですね。

同じプログラムを次のように変えてみます。

```
10 IF INKEY$(1)="Y" THEN PRINT "Yes" ELSE PRINT "No":GOTO 10
```

こうすると，どれかキーが押されるまで，INKEY$関数は何もせずに待ってくれます。ここでは，INKEY$(1)を使うことにしましょう。それから，[Y]が押されて入力される文字が大文字の「Y」であっても小文字の「y」であっても受けつけてくれるように工夫します。ずいぶんと親切で使いやすいプログラムができそうです。

データの入力を終えたいときには，時刻入力のときに[E]キーを押すようにしておきます。こうしておかないと，再度30回分のON/OFFのデータを入力しなければなりません。

●何度でも呼び出しOKのサブルーチン

これまでテレビをコントロールする部分，コードの分解，入力する部分などをばらばらにしてみてきました。それぞれの部分で，必要なステートメントや関数を確かめてきましたから，それら小さな部分の働きはわかりました。みなさんも，これらの小さな部品をまとめて１つのプログラム化をしたいと思っているわけですが，ここで「サブルチーン」という考え方が必要になります。

「サブルーチン」とは，プログラム中で何度も同じ処理をする部分をひとまとめにして，必要に応じていつでも何度でも呼び出して使えるようにしたプログラムの一部をさします。つまり，これまで組み立ててきたプログラムの部分はすべて「サブルーチン」だったわけです。

サブルーチンに対してプログラムの主幹になる部分を「メインルーチン」と呼びます。

サブルーチンを利用するときには，GOSUBという命令を使います。GOSUBの後にはサブルーチンのはじまりの行番号を書きます。これはGOTO文でも同じです。サブルーチンの終わりには，GOSUBで呼び出された部分に戻るために，RETURNという命令を書いておきます。わかりやすいように表わしてみると，イラストのようになります。GOSUBとGOTOとの違いは，GOSUBの場合，いつでもRETURNと組み合わされて使われるのでプログラムの流れがいっきに飛んでも必ず元の場所まで戻ってくるという点にあります。GOSUBの後には行番号を書いてプログラムの飛び先を指示しますが，行番号のかわりにサブルーチンの名前を使うとさらにわかりやすくなります。

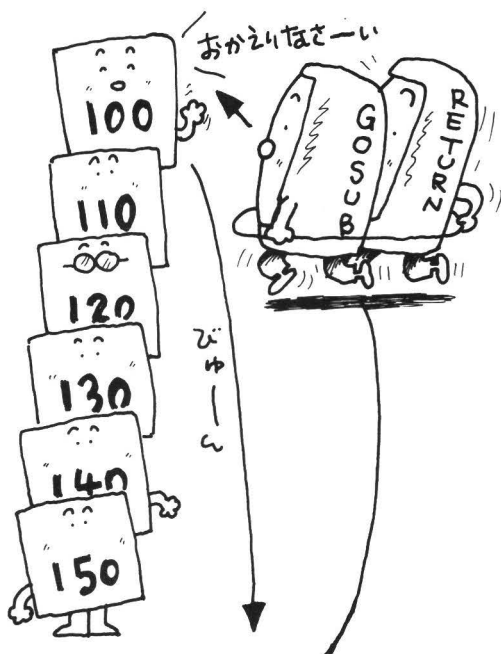
X１では「LABEL」（ラベル）を使ってサブルーチンに名前をつけることができます。LABELというのは名札のことですから，文字どおりラベルを貼ってサブルーチンを整理しておくわけです。LABELの使い方は，

LABEL “ラベル名”

とします。

LABELは，GOSUB文だけではなくGOTO文にも使えます。

さて，これでプログラムの構成が見えてきました。指定した時刻やチャンネルなどのデータの入力部分での仕事が終わったら，実際にテレビをコントロールする部分を繰り返して条件判断を続けていけるようにすればいいわけです。GOSUB文とLABELを使ってプログラムを組みます。

```
10 OPTIONBASE 1:DIM TC$(30)
20 CLS
30 GOSUB "ﾆｭｳﾘｮｸ"
40 GOSUB "ｺﾝﾄﾛｰﾙ"
50 GOTO 40
110 LABEL "ﾆｭｳﾘｮｸ"
```

```
120 I=1
130 INPUT "ナン時 (`E` デ オワリ): ",JI$
140 IF JI$="E" GOTO 380
150 IF VAL(JI$)>23 OR VAL(JI$)<0 OR FRAC(VAL(JI$))<>0 THEN GOTO
160 INPUT "ナン分                    : ",FU$
170 IF VAL(FU$)>59 OR VAL(FU$)<0 OR FRAC(VAL(FU$))<>0 THEN GOTO
180 IF LEN(JI$)=1 THEN JI$="0"+JI$
190 IF LEN(FU$)=1 THEN FU$="0"+FU$
200 JF$=JI$+FU$
210 PRINT "スイッチ ON --- 1   OFF --- 0 : ";
220 SW$=INKEY$(1)
230 IF SW$<>"0" AND SW$<>"1" THEN GOTO 220
240 PRINT SW$
250 IF SW$="0" THEN CHAN$="00":GOTO 290
260 INPUT "CHANNEL=";CHAN$
270 IF VAL(CHAN$)<1 OR VAL(CHAN$)>12 OR FRAC(VAL(CHAN$))<>0 THEN
 GOTO 260
280 IF LEN(CHAN$)=1 THEN CHAN$="0"+CHAN$
290 PRINT JI$;"時";FU$;"分   ";
300 IF SW$="1" THEN PRINT " ON CHANNEL";CHAN$ ELSE PRINT " OFF"
310 PRINT "OK ?   (Y/N) : ";
320 Z$=INKEY$(1)
330 IF Z$<>"N" AND Z$<>"n" AND Z$<>"Y" AND Z$<>"y" THEN GOTO 320
340 PRINT Z$
350 IF Z$="Y" OR Z$="y" THEN TC$(I)=JF$+SW$+CHAN$ ELSE GOTO 130
360 I=I+1
370 GOTO 130
380 EF=I-1
390 RETURN
```

らんだむめも

行を切ったりつなげたり…

非常に大きなプログラムでメモリが不足しそうなときには，マルチステートメントを使ってなるべくたくさんの内容を1行に入れてしまい，行数を減らすとメモリの節約になります。ところで，実に便利なことにX1では，CTRL＋Wという操作で，2行に分かれている行を1行につなげることができるのです。たとえば，

10 CLS
20 TIME=0

の2行をつなげるとしましょう。カーソルを10行に移動させ，CTRL＋Wを押します。10行の中であれば，カーソルはどの位置にあってもかまいません。次にカーソルを20行のTIMEのTに合わせます。この状態でDELキーを押し続けると，T以下が上の行，つまり10行の後に移動してきます。適当なところでキーをはなし，コロンを書き入れてリターンキーを押せば結合完了です。20行はまだ残っていますから，20CRとして20行を消しておかなければならないのはもちろんです。

逆に，行が長すぎたり，複雑になってしまったときは，CTRL＋Jを使って行を2つにわけることもできます。たとえば，今の例で，

10 CLS：TIME=0

となったのを，もとの2つの行にわけるには，カーソルを「CLS」の後のコロンに合わせ，CTRL＋Jを押します。コロン以下が次の行に移りますから，適当にスペースをあけて行番号をつければよいわけです。本当に便利ですね。

```
400 LABEL "ｺﾝﾄﾛｰﾙ"
410 FOR I=1 TO EF
420 JI$=LEFT$(TC$(I),2)
430 FU$=MID$(TC$(I),3,2)
440 SW$=MID$(TC$(I),5,1)
450 CHAN$=RIGHT$(TC$(I),2)
460 IF JI$<>LEFT$(TIME$,2) THEN  GOTO 510
470 IF FU$<>MID$(TIME$,4,2) THEN  GOTO 510
480 IF SW$="0" THEN TVPW OFF:GOTO 520 ELSE TVPW ON
490 CHAN=VAL(CHAN$)
500 CHANNEL CHAN
510 NEXT I
520 RETURN
```

3-7 プログラムに旗が立つ?

●はたして動くか?

「テレビ24時間予約プログラム」もだいぶ完成に近づきました。とりあえずここまで作り上げたプログラムを動かしてみましょう。たくさんのデータを入れるのもたいへんですから，まず一度だけ指定した時刻にテレビのスイッチをONにして試してみることにします。

現在の時刻を確かめるのに時計を見るのもよいですが，ここは1つX1を使って確かめてみましょう。ファンクションキーの2番目には「? TIME$」という命令が入っています。

プログラムを走らせて，1分ほどしてテレビがONになるように指定してみましょう。エラーが出るようでしたら，何度でもプログラムを見直して確かめてください。

努力のかいあって，テレビが映りはじめました。しかし，どうも様子がおかしいようです。音は途切れるし，画面のチャンネル表示もチラつきます。どうしてでしょうか? 原因を調べてみましょう。

テレビをコントロールしているサブルーチンを見てみましょう。一見しておかしなところはないようです。しかし考えてみると，このサブルーチンはメインルーチンの中のGOSUB文によって何度も呼び出されています。TIME$の示す時刻と指定した時刻が一致すれば，電源を入れ，チャンネルを切りかえるという作業をします。ところが，指定した時刻は分の単位までで，秒までは指定していませんから，その一致した時刻の0秒から59秒までの1分間は，何度も電源を入れてチャンネルを切りかえるという操作をしているのです。このために，先ほどのような現象が生じてしまったわけです。

スイッチをONにする作業は，時刻が一致したときに一度だけ行なえばいいのですから，何度も行なうことを避けるため，1つのコードに対して，テレビのコントロールを行なったかどうかのマーク，言いかえれば書類でいう「処理済」のスタンプを押しておけばいいわけです。このスタンプが押してあればテレビをコントロールする必要がなくなるわけです。

この処理済のスタンプを押すスペースを，プログラムにも作ることにします。コードの文字配列（TC$(n)）に対応した数値配列（CH(n)）を用意します。それぞれの配列の添字の同じものを対応させ，テレビのコントロールがまだ行なわれていないものはCH(n)＝1にし，行なわれたものをCH(n)＝0にするようにします。こうしておけば，テレビをコントロールする前に，IF文でCH(n)を1か0かを調べ，0だったらテレビのコントロールを行なわないようにすればいいわけです。

プログラムを手直ししていきましょう。まず配列の宣言をします。

```
10 OPTIONBASE1:DIM TV$(30),CH(30)
```

このように，複数の配列を宣言するときは，カンマで区切って1つのDIM文に収めてしまうことができます。CH(30)ははじめはすべて未処理のわけですから，すべて1にしておきます。そのために“ショキカ”（初期化）というサブルーチンを作っておきます。FOR～NEXTを使って，1から30までの配列に順次1を代入します。

```
10 OPTIONBASE 1:DIM TC$(30),CH(30)
20 GOSUB "ｼｮｷｶ"
30 GOSUB "ﾆｭｳﾘｮｸ"
40 GOSUB "ｺﾝﾄﾛｰﾙ"
50 GOTO 40
60 LABEL "ｼｮｷｶ"
70 FOR I=1 TO 30
80 CH(I)=1
90 NEXT
```

このCH(n)のような変数を一般に「フラグ」と呼びます。フラグとは旗のように他の変数に対して目印になるような働きをうまく表現しています。この目印であるフラグを使うため，コントロールのサブルーチンを次のように書きかえます。

```
410 FOR I=1 TO EF
420 IF CH(I)=0 THEN GOTO 520
430 JI$=LEFT$(TC$(I),2)
440 FU$=MID$(TC$(I),3,2)
450 SW$=MID$(TC$(I),5,1)
460 CHAN$=RIGHT$(TC$(I),2)
470 IF JI$<>LEFT$(TIME$,2) THEN GOTO 520
480 IF FU$<>MID$(TIME$,4,2) THEN GOTO 520
490 IF SW$="0" THEN TVPW OFF:CH(I)=1:GOTO 520 ELSE TVPW ON:CH(I)
=0
500 CHAN=VAL(CHAN$)
510 CHANNEL CHAN
520 NEXT I
530 RETURN
```

配列変数にたくさんのデータを保存しておき，後からのデータを取り出して使う場合，この「フラグ」がたいへん役立つものとなります。このフラグで「テレビ24時間予約プログラム」も最後の仕上げにかかるようになりました。

```
10 OPTIONBASE 1:DIM TC$(30),CH(30)
20 GOSUB "ｼｮｷｶ"
30 GOSUB "ﾆｭｳﾘｮｸ"
40 GOSUB "ｺﾝﾄﾛｰﾙ"
50 GOTO 40
60 LABEL "ｼｮｷｶ"
70 FOR I=1 TO 30
80 CH(I)=1
90 NEXT
100 RETURN
110 LABEL "ﾆｭｳﾘｮｸ"
120 I=1
130 INPUT "ﾅﾝ時 (`E` ﾃﾞ ｵﾜﾘ): ",JI$
140 IF JI$="E" THEN GOTO 380
150 IF VAL(JI$)>23 OR VAL(JI$)<0 OR FRAC(VAL(JI$))<>0 THEN GOTO130
160 INPUT "ﾅﾝ分                : ",FU$
170 IF VAL(FU$)>59 OR VAL(FU$)<0 OR FRAC(VAL(FU$))<>0 THEN GOTO160
180 IF LEN(JI$)=1 THEN JI$="0"+JI$
190 IF LEN(FU$)=1 THEN FU$="0"+FU$
200 JF$=JI$+FU$
210 PRINT "ｽｲｯﾁ ON --- 1   OFF --- 0 : ";
220 SW$=INKEY$(1)
230 IF SW$<>"0" AND SW$<>"1" THEN GOTO 220
240 PRINT SW$
250 IF SW$="0" THEN CHAN$="00":GOTO 290
260 INPUT "CHANNEL=";CHAN$
```

```
270 IF VAL(CHAN$)<1 OR VAL(CHAN$)>12 OR FRAC(VAL(CHAN$))<>0 THEN
 GOTO 260
280 IF LEN(CHAN$)=1  THEN CHAN$="0"+CHAN$
290 PRINT JI$;"時";FU$;"分   ";
300 IF SW$="1" THEN PRINT " ON CHANNEL";CHAN$ ELSE PRINT " OFF"
310 PRINT " OK ?   (Y/N) : ";
320 Z$=INKEY$(1)
330 IF Z$<>"N" AND Z$<>"n" AND Z$<>"Y" AND Z$<>"y" THEN GOTO 320
340 PRINT Z$
350 IF Z$="Y" OR Z$="y" THEN TC$(I)=JF$+SW$+CHAN$ ELSE GOTO 130
360 I=I+1
370 GOTO 130
380 EF=I-1
390 RETURN
400 LABEL "ｺﾝﾄﾛｰﾙ"
410 FOR I=1 TO EF
420 IF CH(I)=0 THEN GOTO 520
430 JI$=LEFT$(TC$(I),2)
440 FU$=MID$(TC$(I),3,2)
450 SW$=MID$(TC$(I),5,1)
460 CHAN$=RIGHT$(TC$(I),2)
470 IF JI$<>LEFT$(TIME$,2) THEN GOTO 520
480 IF FU$<>MID$(TIME$,4,2) THEN GOTO 520
490 IF SW$="0" THEN TVPW OFF:CH(I)=1:GOTO 520 ELSE TVPW ON:CH(I)
=0
500 CHAN=VAL(CHAN$)
510 CHANNEL CHAN
520 NEXT I
530 RETURN
```

3-8 画面を整理

●X 1，画面の構成

さてここまで「テレビ24時間予約プログラム」をだんだんと成長させ組み上げてきました。

BASICの命令で24時間テレビのコントロールを行なうという最初の目的をどうやら達しましたし，このままでも十分使うことができるプログラムです。ただ，チャンネルや時刻を入力するにつれて次々に行が変わってしまい，画面が非常に見苦しくなってしまうのが気になります。ここでは，画面の表示を整理し，不十分なところも手を入れてしっかりしたプログラムにまとめ上げたいと思います。

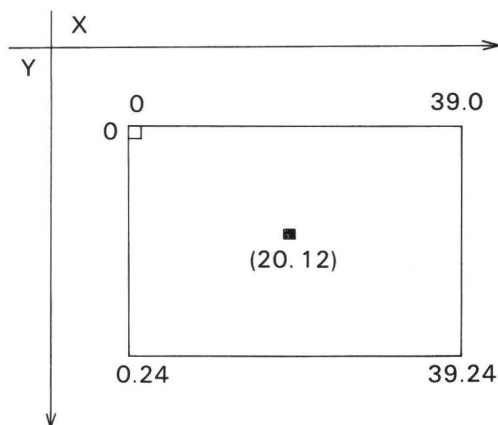
時刻やON/OFF，それにチャンネルなどのデータの入力は右表のような形に整理できるとよいですね。1回分のデータを1行で表わせるようにしておくと，簡潔でわかりやすく誤りなども見つけやすくなるでしょうから，最初に画面表示のレイアウトを決めてしまいましょう。

時刻	ON/OFF	CHANNEL
23：30	ON	12
23：35	OFF	⋮
⋮	⋮	⋮
⋮	⋮	⋮

X 1の画面の構成は，横40文字×縦25行，または，横80文字×縦25行のどちらかを選ぶことができます。この画面構成を文字数から40字モード，80字モードというように呼びます。みなさんのX 1は，今40文字モードになっていると思いますから，これを80文字モードにしてみましょう。

```
WIDTH 80⏎
```

とします。画面のOKの文字もカーソルも，横幅が半分位になったはずです。試しに，リストをとってみると，その違いがよくわかります。40文字モードは，画面の見やすさを重視する場合に使い，80文字モードは，なるべくたくさんの文字を表示したいときや，ビジネスでの表作りなどのときに多く使われます。

INPUT文でデータを入れてリターンキーを押すと，行が変わってしまいます。時刻やチャンネルなどのデータを整然と表示するには，画面上の特定の位置を指定してINPUT文などの命令を実行させる必要があります。

X 1の画面は，40文字モードの場合，文字を表示できる場所（座標）は図のようになっています。横は0から39までの40文字，縦は0から24までの25行になります。この何文字目の何文字目というように座標で表示位置を指定してやれば，きちんと整理された画面になります。表示の位置を指定するのがLOCATE文です。

●座標を決める

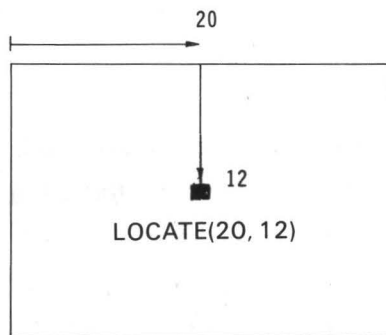
LOCATE文は,

```
LOCATE x, y
```

という形で使い，Xは横方向，Yは縦方向の数値を書きます。つまり，座標を指定します。Xは，40文字モードの場合0から39，80文字モードの場合は0から79の範囲になります。Yはどちらのモードの場合でも0から24までです。この範囲外を指定するとエラーになります。

40文字モードでは，画面の中央の位置は，LOCATE 20, 12になります。LOCATE文ではPRINT文で画面に表示される文字や記号の位置を指定するほかに，INPUT文で使われるプロンプト文の位置やINKEY$関数のカーソルの表示を決めることもできます。たとえば，画面の中央に♥を表示するのであれば，

```
LOCATE 20,12:PRINT"♥"
```

となります。

X1のBASICにはCURSOR文というものもありますが，働きはLOCATE文とまったく同じです。試してみてください。

●プログラムの表示を整理

このLOCATE文を使い，テレビコントロールの1回分のデータを1行にまとめてみます。INPUT文やINKEY$関数のカーソルの位置を，Yの値はそのままにXの値だけを変えていけばよさそうです。1回分のデータを入力し終わったなら，Yの値に1を加えて，またXの値だけを変えていきます。しかし，この方法ですと，誤ったデータを入力してしまって，IF文によって再度入力し直すときに，前のデータが消えずに残ってしまいます。文字を消すには，その文字の位置にスペースをPRINT文で表示するという方法がよく使われますが，たくさんの文字が書かれている場合などは，この方法ですと消し切れなくなってしまうこともあります。

そのようなときは，カーソルから後を1行分そっくり消してしまうことができれば便利です。先に出てきた[CTRL]＋[E]を使うとそのようにできます。このキーと同じ働きをして，プログラム中に入れることができるのがCHR$(5)です。CHR$は前にも少し書いたとおり，数値をアスキーコードとみなして，そのコードに対応する文字などに変換される関数です。数値は0から255までで，たとえばCHR$(65)はAという文字になります。ところが，0から31までは文字や記号ではなく，**制御コード**というものに割りあてられています。CHR$(7)は，[CTRL]＋[G]と同じ働きをするもので，いろいろな働きをプログラム中に書き入れることができます（巻末アスキーコード表参照)。

●コンピュータは巻き物

このプログラムでは，最初に画面に表示させる見出し（タイトル）を書くようにします。ところが，データを入力する回数が多くなって一番下の行になった場合，画面全体が上に１行上がっていきます。すると，このタイトル部が画面から出て見えなくなってしまいます。このような動作を「スクロールアップ」（スクロールとは，くるくると巻かれたもののこと）といいます。タイトル部を画面上に残しておくには，スクロールする部分からタイトルをはずしておけば良いわけです。スクロールする範囲（スクロールエリア）を決めるには，CONSOLE文を使います。

CONSOLE Ys,Yn,Xs,Xn

普通CONSOLE文による指定は縦方向（行数）だけですが，Ｘ１では横方向にも指定できます。Ysはスクロールのはじまる最初の行，Ynはスクロールする行数を指定します。Ｘ方向も同じように指定し，縦方向だけの指定ならば横方向のXs, Xnは省略することができます。横方向だけの指定のときは必ず縦方向も指定しないとエラーになりますから注意してください。

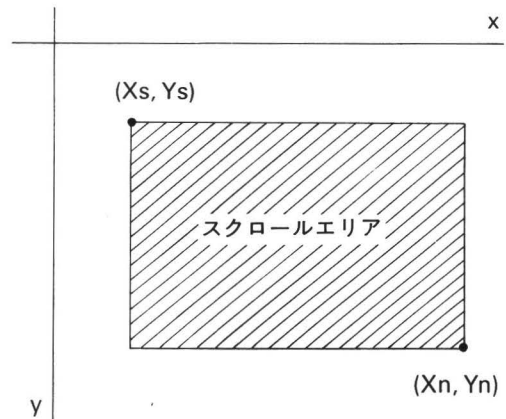
次のプログラムを実行してみると，CONSOLE文の働きがよくわかると思います。

```
10 WIDTH40:CLS
20 A$="SHARP PARSONAL COMPUTER [ﾊﾟｿｺﾝ ﾃﾚﾋﾞ X1]"
30 FOR YS=0 TO 24:CONSOLE YS,25-YS
40 FOR I=1 TO 25-YS:PRINT A$:NEXT I
50 CONSOLE 0,25:CLS:NEXT YS
60 FOR YN=25 TO 1 STEP-1:CONSOLE 0,YN
70 FOR I=1 TO YN:PRINT A$:NEXT I
80 PRINT A$
90 NEXT YN
100 FOR XS=0 TO 39:CONSOLE 0,25,XS,40-XS
110 PRINT A$:PAUSE 2:CLS
120 NEXT XS
130 CONSOLE 0,25
```

110行にあるPAUSEは，プログラムの実行を一時休止させるもので，数値１単位につき約0.1秒休止し，次に進んでいきます。

スクロールエリア外に，LOCATE文で位置を指定することはできません。また，一度指定されたスクロールエリアは，改めて指定し直されるまで変わることはありませんが，WIDTH文で文字数を変えたときや，[CTRL]＋[D]を押したときには解除されて，CONSOLE 0, 25, 0, 39(79)の状態に戻ります。

せっかく表のように表示させるのですから，きれいにスタートさせましょう。[SHIFT]＋[CLR]か，[CTRL]＋[L]を押すと画面がきれいになりました。

プログラム中でこれを行なうのは，CHR$(12)の他にCLSがあります。CLSはClear Screenの略ですから，文字どおり画面をきれいにする命令です。スクロールエリアが指定されている場

合，CLSが働くのはそのエリアに限られてます。

さて，画面の見出しの部分を０行から２行までの計３行にわたって書きます。160行にあるPRINT STRING$(39, "―")によって，データ表示部と分けます。STRING$は，任意の文字列を任意の数だけ用意するものです。そしてCONSOLE文で４行目から25行目までをスクロールエリアに指定します。

TC$(１)の基になるデータが入力される画面上の行は，Ｉ＝１のとき第３行，Ｉ＝２のときには第４行となりますので，LOCATE文のＹの値にはＩ＋２をあてはめていけばよいでしょう。Ｙの値にはＩの代りにＳという変数を使っています。これは，データが20回分入力されると，スクロールエリアがデータでいっぱいになってしまうので，20回分入力し終わったところで，一度CLS文で画面上の表示を消してから，再度第３行から入力を行なうようにします。この判断は380行で行ないますが，Ｉの値は再び１に戻すことはできないので，Ｉの代りにＳを使うわけです。LOCATE文やCONSOLE文を使うときは，画面上の１文字目，または１行目は０になることを忘れないでください。20行目にあるINITについては《8 - 6》で説明しますのでそちらを参照してください。

これで「テレビ24時間予約プログラム」が完成しました。はじめてのプログラムにしては多すぎるほどの内容を持っていますが，わからない個所は何度でも読み返して理解しておくようにしてください。

```
10 OPTIONBASE 1:DIM TC$(30),CH(30)
20 INIT:WIDTH40:CLS
30 GOSUB "ｼｮｷｶ"
40 GOSUB "ﾆｭｳﾘｮｸ"
50 LOCATE 16,12:PRINT TIME$
60 GOSUB "ｺﾝﾄﾛｰﾙ"
70 GOTO 50
80 LABEL "ｼｮｷｶ"
90 FOR I=1 TO 30
100 CH(I)=1
110 NEXT
120 RETURN
130 LABEL "ﾆｭｳﾘｮｸ"
140 CLS:LOCATE 0,0:PRINT "時時:分分      ON / OFF      CHANNEL      OK
?"
150 LOCATE 0,1:     PRINT "E=ｵﾜﾘ        1     0         1-12        (Y/N
)"
160 LOCATE 0,2:PRINT STRING$(39,"―"):CONSOLE 3,22
170 I=1:S=1
180 LOCATE 0,S+2:INPUT "",JI$
190 IF JI$="E" GOTO 410
200 IF VAL(JI$)>23 OR VAL(JI$)<0 OR FRAC(VAL(JI$))<>0 THEN LOCAT
E 0,S+2:PRINT CHR$(5):GOTO 180
210 LOCATE 2,S+2:PRINT ":"
220 LOCATE 3,S+2:INPUT "",FU$
230 IF VAL(FU$)>59 OR VAL(FU$)<0 OR FRAC(VAL(FU$))<>0 THEN LOCAT
E 3,S+2:PRINT CHR$(5):GOTO 220
240 IF LEN(JI$)=1 THEN JI$="0"+JI$
250 IF LEN(FU$)=1 THEN FU$="0"+FU$
260 JF$=JI$+FU$
270 LOCATE 13,S+2
280 SW$=INKEY$(1)
290 IF SW$<>"0" AND SW$<>"1" THEN GOTO 280
300 IF SW$="0" THEN CHAN$="00":LOCATE 13,S+2:PRINT "OFF":GOTO 34
0 ELSE LOCATE 13,S+2:PRINT "ON"
```

```
310 LOCATE 25,S+2:INPUT "",CHAN$
320 IF VAL(CHAN$)>12 OR VAL(CHAN$)<1 OR FRAC(VAL(CHAN$))<>0 THEN
 LOCATE 25,S+2:PRINT CHR$(5):GOTO 310
330 IF LEN(CHAN$)=1 THEN CHAN$="0"+CHAN$
340 LOCATE 36,S+2
350 Z$=INKEY$(1)
360 IF Z$<>"Y" AND Z$<>"y" AND Z$<>"N" AND Z$<>"n" THEN GOTO 350
370 PRINT:IF Z$="Y" OR Z$="y" THEN TC$(I)=JF$+SW$+CHAN$:LOCATE 3
6,S+2:PRINT "*"  ELSE 180
380 I=I+1:S=S+1:IF S=21 THEN CLS:S=1
390 IF I=30 THEN GOTO 50
400 GOTO 180
410 FE=I-1
420 RETURN
430 LABEL "ｺﾝﾄﾛｰﾙ"
440 FOR I=1 TO FE
450 IF CH(I)=0 GOTO 550
460 JI$=LEFT$(Tc$(I),2)
470 FU$=MID$(Tc$(I),3,2)
480 SW$=MID$(Tc$(I),5,1)
490 CHAN$=RIGHT$(Tc$(I),2)
500 IF JI$<>LEFT$(TIME$,2) GOTO 550
510 IF FU$<>MID$(TIME$,4,2) GOTO 550
520 IF SW$="0" THEN TVPW OFF:CH(I)=0:GOTO 550 ELSE TVPW ON:CH(I)
=0
530 CHAN=VAL(CHAN$)
540 CHANNEL CHAN
550 NEXT I
560 RETURN
```

3-9 プログラムを保存

●テープにセーブ

みなさんがキーボードから打ち込んだプログラムは，コンピュータ内部のメモリに記憶されています。

このプログラムは，メモリに記憶されている限り，いつでも必要なときにRUNという命令によって走らせることができます。

人間には，ときどきうっかり忘れてしまったり，時がたつにつれて記憶が薄れたりということがありますが，一晩眠ったからといって前日のことをすっかり忘れてしまうということはありません。ところが，コンピュータは，せっかく苦労して打ち込んだプログラムも，電源を切った瞬間にすべて忘れてしまいます。だからといって，寝るときも外出するときもＸ１の電源を入れっぱなしにしておくわけにはいきません。そこで，メモリの中のプログラムを別の形にして記録しておかなくてはなりません。

ここでは，カセットテープへのプログラムの記録を行なってみましょう。Ｘ１の本体にはカセットデッキがついています。

プログラムをカセットテープに記録するには，SAVEという命令を使います。また，このようにしてメモリの中のプログラムを外部記憶装置で記録することを「セーブする」といいます。

SAVE "TV-24" ↵

とすると，Ｘ１本体のインジケータランプのWRITEがオレンジ色に点灯して，自動的にテープが回りはじめます。プログラムの記録がはじまりました。もちろん，カセットデッキの中にはプログラムを記録するためのカセットテープがセットされていなければなりません。カセットテープが入っていないときは，

Out of tape

というＸ１からメッセージが画面に現われます。このようなときには，カセットテープをセットしてもう一度同じようにSAVE命令をキーボードから打ち込んでやらなくとも，カーソルを，前に記入したSAVE命令のところまで移動させてリターンキーを押すだけでよいのです。もし「Out of tape」のメッセージが目ざわりであれば，その行の一番左の端までカーソルを移動させ CTRL キーと Z キーをいっしょに押してみましょう。CTRL キー Z キーを同時に押すとその行のカーソルから右側の文字，それにその下の行の全部がいっぺんに消えてしまいます。

さて，カセットテープが止まり画面に「Ok」と表示されたら，テープへのプログラムの記録は完了です。正しく記録されているかどうかを確かめておきましょう。

●セーブをチェック

まず，プログラムの記録を開始したところまでカセットテープを巻き戻さなければなりません。新しいカセットテープを使ってテープの最初から記録をはじめた場合には，巻き戻しボタンを押すだけでよいのですが，さてカセットテープの途中に記録を行なった場合はどうすれば

よいでしょう。カセットデッキの横にあるカウンタの数字を目安にするのは確かに良い方法です。しかし，X1のキーボードにはわざわざカセットコントロールキーが用意されているのですから，これを使いましょう。

SHIFT キーと［◀◀］キーをいっしょに押します。

APSS-1と命令を与えてもX1は同じ動作をします。こうすると，X1は今書き込んだプログラムが記録されている最初の部分まで，カセットテープを巻き戻してくれます。前に勉強したことですが，思い出しましたか？

カセットテープにプログラムが正しく記録されているかどうかを確認するための命令は2つあります。

VERIFY "TV-24" ↵
LOAD? "TV-24" ↵

どこが違うかといえば，実はどこにも違いはなく2つの命令はまったく同じ働きをします。ですからどちらを使っても構わないのですが，LOAD?命令ではクウェスチョンマークをつけ忘れるだけで別の命令となってしまいます。SAVEとは逆にカセットテープからプログラムを読み出す命令がLOADです。セーブしたプログラムに誤りがなければよいのですが，万一正しくセーブされていなかった場合，LOAD命令を実行するとX1のメモリの中のプログラムまでこわれてしまうことになります。誤りを避けるためには，VERIFYを使う方がよいでしょう。

VERIFY，またはLOAD?命令を実行すると，X1はカセットテープに記録されたプログラムと，メモリ内のプログラムを1文字ずつ比べ，間違いがなければ「OK」と表示し，1カ所でも違うところがあれば「Tape read error」と表示します。「Tape read error」が表示されたときには，もう一度セーブをやり直さなければなりません。このようにして，プログラムのセーブが正しくできたかどうかを確認することを，「ヴェリファイする」といいます。これは大切な作業です。

「"」（ダブルクォーテーション）で囲った「TV-24」というのは，プログラムの名前です。プログラムの名前のことを一般にファイル名と呼びます。ダブルクォーテーションの間には13文字までの記号や文字を自由に書けますから，みなさんで工夫してプログラムにも素敵な名前をつけてください。

プログラムを記録したカセットテープは，ミュージックテープと同じように保管します。ファイル名とカセットデッキのカウンタの数字，記録した年月日などをカセットテープのラベルに記入します。ミュージックテープのラベルにアーチストの名前を記入したり，録音年月日を記入するのと同じです。直射日光を避けて，ホコリの少ないところで保管するという点も同じです。テープには磁気によってプログラムが記録されていますから，強い磁気の近くでの保管もいけません。ミュージックテープでは音が多少変になったという程度の事故が，パソコンでは致命的であることもしばしばです。

最近では，パソコン用として10分，15分といった短いテープも市販されています。プログラムの保管には60分，90分といった長い時間のカセットテープよりも使い勝手がよいので，こちらをおすすめします。これらパソコン用テープにはリーダーテープ（テープの最初につけられた録音できない部分）がついていませんから，プログラムの記録にはもってこいです。

ファイルネームの後をピリオドで区切って3文字つけることができます。これは，プログラ

ムを修正してセーブし直すときなど，同じファイルネームを区別するためなどに使うことができます。これを拡張子といいます。

らんだむめも

テープ内容をみたいときは…

BASICの勉強が進んでくるにつれて，プログラムを記録したカセットテープの数もだんだんと増えていきます。新しいプログラムを作りはじめたとき，以前に作ったプログラムを参考にしたいということもあるでしょう。こんなときに，さてどのテープに何のプログラムが入っていたかな，と探しまわらなくてもいいように，プログラムテープは上手に整理しておきたいものです。そのためには，テープへの記録にも気を使っておくとあとあと便利だということは，前にもお話したとおりです。どんなプログラムが入っているのかをラベルにきちんと記録しておくのがよいということもお話しました。もしラベルに何も書かれていないテープや，ラベルがなくなってしまったテープがあったら，

LFILES

と命令します。X１は，カセットテープを一度全部巻き戻してから順に早送りをして，プログラムの名前（ファイル名）だけを拾い出します。しかも驚いたことに，そのプログラムを記録した年月日や時刻もファイル名とともに画面に表示してくれます。FILESの命令を実行しても，現在X１のメモリの中に入っているプログラムにはなんの影響もありません。ファイル名がわかったら，今度はきちんと記録して，カセットテープを保管しましょう。

プリンタを持っている人は，

LFILES

で紙の上に打ち出すことができますから，これをテープのケースに入れておくとよいでしょう。

3-10 文法が違ってるよ!!

●コンピュータが間違いを教えてくれる

ここまで，1つのプログラムを少しずつ打ち込んできました。慣れないキーボードに向かって悪戦苦闘した方も多かったのではないでしょうか。

あげくのはてに，RUNさせたら画面には何やら「×× error」のメッセージ。エラーというからには，どこかが間違っているに違いないのに，はじめはどこがどう間違っているかさっぱりわからないという方もいたことでしょう。プログラムの誤りをバグとか虫と呼びます。また，誤りを直す作業をデバッグ，あるいは虫を取るといいます。エラーと表示されてプログラムの実行が止まってしまったときに能率よくデバッグする方法を紹介しましょう。

この「×× error in ○○」というコンピュータからのメッセージを「エラーメッセージ」といいます。「in ○○」はそのエラーが起こった行番号です。BASICでは，このように誤りがあればその誤りの原因と，発生した行番号を知らせてくれます。

コンピュータの動作を直接コントロールする機械語（コンピュータの中で扱われている言語）では，誤りを知らせてくれるどころか，誤った部分は誤ったとおりに，そのまま実行されてしまいます。コンピュータの動作が，キーボードからまったくコントロール不能の状態（暴走状態）にまで及んでしまうこともあります。こうなってしまったらもうそのプログラム自体も破壊されてしまうことが多く，せっかく打ち込んだプログラムもたった1文字の誤りでだいなしになってしまいます。BASICでは，誤りがあればそこでプログラムの実行を止め，エラーメッセージを出してプログラムの修正を要求してきます。このように，コンピュータと人間が対話する形を持っていることから，BASICのようなコンピュータ言語を「対話型言語」とも呼びます。

●文法の間違いだよ！

エラーには，いくつもの種類がありますが，たぶんみなさんがもっとも多く経験されたのは，「Syntax error」だったのではないでしょうか。

Syntaxというのは文法のことです。Syntax errorは，「あなたの打ち込んだプログラムの内容が，

BASICで決められた文法と合わない」ということを表わしています。BASICでは，たとえばPRINTという命令があれば，これを解釈して画面に表示するために必要な処理を行ないます。ところがこの命令文がたとえ１文字でも間違っていると，コンピュータは何をしてよいかわからなくなります。私たち人間なら，たとえば「PRINT」が「PRIMT」になっていたとしても，前後の関係から，その意味を理解することができます。ところがコンピュータでは，このような融通はききませんから，文法が違っています，というメッセージを出して止まってしまうわけです。正しいプログラムとデータを与えれば，間違えることもなく，途中であきることもなく仕事をするコンピュータですが，逆にいえば融通がきかず非常に頑固でもあるのです。

「Syntax error in ××」と表示されたときのプログラムの直し方を見てみましょう。まず，エラーの出た行を表示させるわけですが，このときには，LIST 行番号とするかわりに，

　LIST. またはL ..

とします。Ｘ１のBASICでは，LISTの最後につけた「.」（ピリオド）に現在実行している行番号を管理する働きがあります。エラーが発生してプログラムが止まってしまった場合には，エラーの発生した行番号がピリオドの中に入っています（といっても，Ａ＝１のような代入文とは違います)。たとえば，次の例では，

```
ナン時 (′E′ デ オワリ) 20
Syntax error in 540
OK
LIST.
540 INPIT"ナン分                ",FU$
OK
■
```

540行にエラーがありますから，「LIST.」とすると，その行が表示されます。ここでは，「INPUT」とすべきところを「INPIT」と打ち間違えたようです。このようなときは，カーソルを違っている「Ｉ」に重ね，正しい「Ｕ」に直します。そしてリターンキーを押せば修正完了です。

さらに便利なコマンドに「EDIT」があります。EDITは編集の意味ですが，BASICでは１行の中身の編集を行ないます。先ほどの場合，540行にエラーがあり，それで修正しようとしているわけですから，EDIT 540 または E. 540 とするか，先ほどのピリオドの働きを利用して，E.. としてやると，次の例のようにその行が表示され，カーソルはその先頭の命令のところに現われます。そのまま間違えたところにカーソルを移動させて修正を行なえばよいわけです。他にも方法があるように思えますが，実用上は，エラーが出たら，「E.. ⏎」と覚えておけば能率よく修正することができます。

```
ナン時 (′E′ デ オワリ) 20
Syntax error in 540
OK
E..
540 INPIT"ナン分                ",FU$
OK
■
```

ときとして，命令文のつづりの文字がぬけて足りない場合もあります。このようなときは，INST キーを使って足りない文字数だけスペースを空けてから不足分を打ち込めばよいわけです。しかし，次の例のように文字数が多いときや，「：」（コロン）でつないだマルチステートメントの中の１つの命令がそっくり落ちているような場合には，別の方法を使った方が便利です。新しく挿入する命令や文の次の文字にカーソルを合わせておいて，CTRL ＋ A を押します。

```
RUN
Syntax error in 10
OK
E..
10 OPTIOBASE 1:DIM TV$(30),CH(30)
```

キーを押しただけでは，画面上に変化はありません。しかし，ぬけ落ちていた個所をそのままキーインすると，自動的にカーソルから後を右にずらしながら打ち込んだ文字を挿入していきます。つまり「インサートモード」になっているのです。

```
RUN
Syntax error in 10
OK
E..
```

インサートモードは，もう一度 CTRL ＋ A を押すか，カーソルキーでカーソルを移動すると解除されます。

打ち込んだプログラムが大きくなってくると一度でうまく動くということはなかなかありませんが，なるべくならあまりエラーで止まってもらいたくないものです。プログラムを打ち込むときは，速さよりも正確さを大事にして正しい文を打ち込むようにしましょう。

らんだむめも

X１ ふたつの命令文の謎

X１のBASICマニュアルを見ていると，同じ内容を表わす命令語が２つ用意されているものがあることに気がつきます。たとえば，マニュアルの24ページを開いてみると，CURSORとLOCATEというステートメントがあり，説明を読むと，この２つは同じ命令である，と書いてあります。なぜこのようなものがあるのでしょうか。その理由は，BASICに「方言」があるためなのです。

X１を開発したSHARPには，長い歴史を持った「MZシリーズ」というX１の兄貴分にあたるパソコンがあります。これらのパソコンには，優秀な実行速度を備えた純国産の「SHARP BASIC」が多く使われています。それに対し，現在国内で販売されているパソコンのほとんどが，アメリカの「マイクロソフト社」が作ったBASICに，各機種によっていくつかの機能をつけ加えたものを使用しています。この２種類のBASICでは，同じ仕事を表わす命令文が先ほどの例のように異なっているものがあります。X１のBASICでは今までどちらのBASICに親しんできた人でも使えるよう，両方の命令語を用意しているのです。

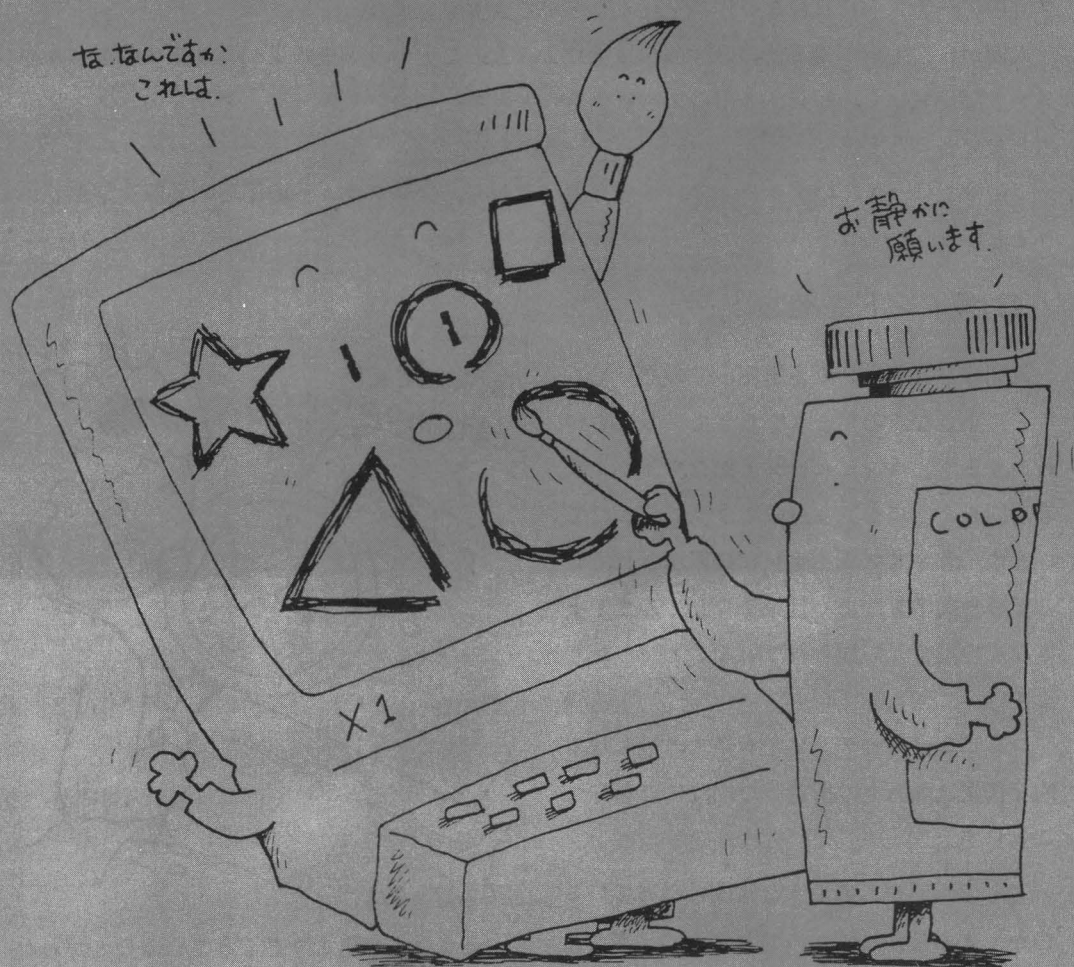
２つの書き方を持つものについて表にまとめておきましたので参考にしてください。

命令語の比較

SHARP BASIC	マイクロソフト系
LIMIT	CLEAR ㊟
VERIFY	LOAD ?
CLR	CLEAR ㊟
（INPUTはX1特有）	LINE INPUT
GRAPH （MZ系とはパラメータの書き方が違う）	SCREEN
CURSOR	LOCATE
CONSOLE （パラメータの書き方に注意）	CONSOLE
LINE （同上）	LINE
DEF KEY	KEY
KLIST	KEY LIST
MUSIC	PLAY
TEMPO	PLAY
LOG(X)	LOG(X)
SHARP BASICのLOG(X)は常用対数であるのに対し，X１ではLOGは自然対数（SHARP BASICのLN（X）である）	
SIZE	FRE(X)

㊟としてあるものについては本書の該当するページ，またはマニュアルを参照してください。

4

X1おもしろグラフィックス

4-1 コンピュータ・グラフィックスってどんなもの?

近頃はテレビのコマーシャルなどでもコンピュータ・グラフィックスがずいぶんと使われるようになってきました。

グラフィックスといえば，絵画やグラフの技法のことです。また絵画やグラフそのもののことも指します。

テレビ・コマーシャルで使われるコンピュータ・グラフィックスは，専用の大型機で制作されていますから，非常に緻密なうえ，色も豊富で鮮明な作品になっています。パソコンで同じ作品を作るのはちょっとむずかしいでしょう。しかし，X１のグラフィックス機能は，他のパソコンに比べると極めて充実しており，しかも手軽に扱えるので，工夫しだいでみなさんにもプロのアーチストに負けない作品を作ることができるでしょう。

人間が日常生活で得る情報の８割が視覚によるものだという調査結果があります。ゲームやアートだけでなく，ビジネス用のプログラムにもおおいにコンピュータ・グラフィックスを応用したいところです。

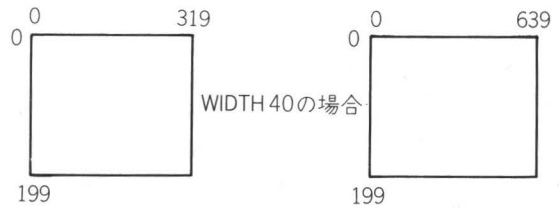
この章では，X１のグラフィック機能を調べながら，プログラムへの応用の実際を学んでいきましょう。

●ポイントをセット！

新聞の写真を虫メガネを使ってのぞいてみると，小さな点の集まりでできていることがわかります。コンピュータ・グラフィックスも，同じように小さな点の集まりで描かれています。ディスプレイテレビに表示されている記号や文字も，よく目をこらしてみると小さな点の集まりで作られていることがわかります。

コンピュータ・グラフィックスの最も基本的な作業は画面上の指定した位置に点を打つことです。

X１の画面がどのような構成になっているのか，まず最初に確かめておきましょう。Xると，小さな点の集まりでできていることがわか１では，前にも少しふれましたが，画面の構成を大まかに分けても２種類指定することができます。

横40文字×縦25行と80文字×25行の2とおり，つまり，40字モードと80字モードです。40字モードの場合，グラフィックスでは，横320×縦200の点を扱うことができます。80字モードのときは640×200です。グラフィックスを描く点は文字と同じようにそれぞれの座標をもっています。LOCATE文を使って，画面上の横X文字目が縦Y文字目というように座標を指定することができました。まったく同じように縦横の座標を指定して，画面上の希望の位置に点を打つことができます（図4-1）。試してみましょう。

〔図4-1〕

```
10 WIDTH 40
20 SCREEN 0,0
30 CLS 4
40 X=160:Y=100
50 PSET (X,Y)
```

画面のほぼ中央に点が打たれるはずです。PSETというのが，点を打つ命令です。今度は色も決めてみましょう。50行を，

PSET (X, Y, 4)

と変えてみると，先ほどと同じ位置に緑色の点が打たれます。PSETは，「ポイントセット」の略で，（　）の中は，X座標，Y座標，色コードとなっています（表4-1）。

表4-1　色コード表

コード	色
0	黒
1	青
2	赤
3	マゼンタ
4	緑
5	シアン
6	横
7	白

10行は，横40字のモードを指定しています。20行のSCREEN文は，グラフィック画面の表示モードを決めるものです。詳しくは後で述べます。30行のCLS文は今まで書かれていた文字やグラフィックスを消してしまう命令です。

この例のようにCLS文の後に0から4までのオペランド（演算数）を指定することができるのですが，このオペランドはSCREEN文と関係が深いので，後でSCREEN文とあわせて説明しましょう。

40行で横と縦の座標を指定し，50行で点（ドット）を打っています。

今度はこの点を消してみましょう。ダイレクトモードで，

PRESET (160, 100, 4)

とすると，先ほど打たれたドットが消されます。PRESETは，ポイントリセットの略で，PSETと同じように，横，縦，色の順で指定します。ある色でPSETを行なったドットは同じ色でPRESETします。

画面が細かい点の集まりで構成されていますから，線を表現するには点を続けて並べればよいわけです。FOR～TO～NEXTを使って，座標を変化させればよいはずです。画面の中央に左から右に線が引かれていきます。

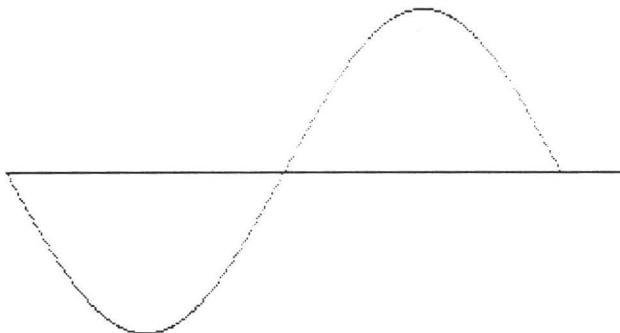
〔プログラム4-1〕

```
10 WIDTH40
20 SCREEN 0,0
30 CLS 4
40 FOR X=0 TO 319
50 PSET (X,100,7)
60 NEXT X
```

このようにして，ドットをセットすることにより，直線や曲線も表示させることができます。SINやCOSの三角関数のグラフを描くこともできます。例としてSIN曲線を描くプログラムを見てみましょう。

〔プログラム4-2〕

```
10 WIDTH40
20 SCREEN 0,0
30 CLS 4
40 FOR X=0 TO 319
50 PSET (X,100,7)
60 NEXT X
70 FOR R=0 TO 360
80 TH=RAD(R)
90 Y=SIN(TH)*80+100
100 X=R*.8
110 PSET(X,Y,4)
120 NEXT R
```

〔画面4-1〕

60行までは〔プログラム4-1〕とまったく同じで，70行からサインカーブを描く部分です。角度を変数Rとして，0から360まで変化させています。BASICでは角度はすべてラジアン(radian)で表現しなければなりません。ラジアンというのは，「円の半径に等しい孤に対する中心角の大きさを表わす角度の単位」のことで，度単位の数式を $\frac{\pi}{180}$ 倍するとラジアン単位に変換できます。このプログラムでは，80行でRAD関数を使っています。RAD関数でラジアンに変換した値を変数THに入れておきます。90行では，ラジアン単位に変換した角度のサインを求め，これを80倍したうえ100を足しています。このサインの値が0から1の間の小さいものなので80倍しているわけです。この補正により，角度0のときのYの値は100となります。前に描いておいた横線のY座標が100ですから，それを原点としたサインカーブを描くことができます(画面4-1)。このような数値の補正は，グラフィックスを扱うときに，しばしば行ないますから，よく理解しておいてください。

4-2 線，いろ，色

●直線を描く，長方形を描く

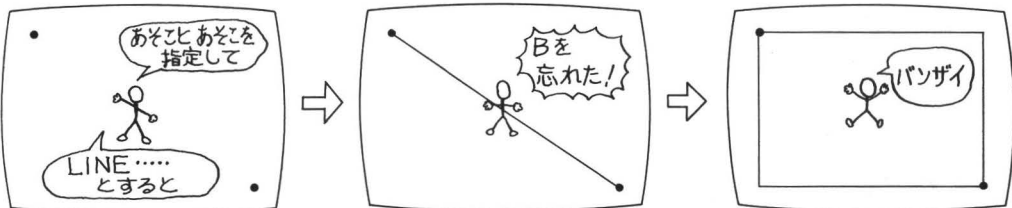
点をつなぐことによって直線や曲線が描けることがわかりました。X 1のBASICには，線を引いたり，線をつないで枠を描く便利な命令があるので，今度はこれを見てみましょう。
〔プログラム 4 - 1〕で引いたのと同じ線を引くには，

とします。LINE文は線を引くための命令です。この命令の後には2つのカッコが続きます。前のカッコの中が線を引きはじめる点のX，Yの座標，後のカッコの中が引き終わりの座標です。つまり，2つの点を指定することによってその2点を結ぶ直線を描くことができるわけです。PSET文では「点を打っていく状態(モード)で」という意味を表わします。最後に書かれた7という数字は白い色を表わします。〔プログラム 4 - 1〕では3行にわたっていた作業が，たった1つの命令ですんでしまうのですから，LINE 文は実に便利な命令といえます。

LINE文は，他にもいろいろな機能を持っています。今度は，画面いっぱいの枠をLINE文を使って描いてみましょう。

LINE (0, 0) — (319, 199), PSET, 7, B

最後のBは「BOX」の略でつまり箱のことです。Bをつけておくと，指定した2点を結んだ線を対角線とする長方形を描くことができます。

〔図 4 - 2〕

さらに，BFとすると，この長方形を描いてその内側を塗りつぶすことができます。

LINE (140, 80) — (180, 120), PSET, 4, BF

とすると，画面の中央に，緑色の正方形が描かれます。いろいろな座標を指定して，どんな位置にどんな線が描かれるか試してみてください。

●破線を描く

次に，ひと続きの実線ではなく，破線や鎖線を描いてみましょう。そのためには，色コード

またはBの指定の後で，さらに線の種類を指定します。線の種類の指定は16進数で行ないます。16進数での指定によりいろいろな線を描くことができますが，ここでは破線と1点鎖線の描き方を覚えておいてください。次の例は，7つの色でそれぞれ実線，破線，1点鎖線を描きます。

〔プログラム4-3〕

```
10 SCREEN0,0:CLS4
20 FOR I=0 TO 7
30 LINE(0,I*8)-(319,I*8),PSET,I ←―――――――実線
40 LINE(0,I*8+2)-(319,I*8+2),PSET,I,&HFFFC ←―破線
50 LINE(0,I*8+4)-(319,I*8+4),PSET,I,&HFFCC ←―鎖線
60 NEXT
```

破線は&HFFFC，1点鎖線は&HFFCCと指定します。このような16進数での指定を**ラインパターン**と呼びます。ラインパターンを省略した場合には実線（つまり&HFFFF）が描かれるのは，これまでの例でおわかりのことでしょう。

LINE文にはPSETとPRESETの他にもう1つXORというモードがあります。XORは排他的論理和といい，前に勉強した論理和のOR，論理積のANDの仲間です。これらの仲間を論理演算子といいます。X1のグラフィックスでは，LINE文にXORを指定して線を引くと，前に表示されていた線の色と，新しい線の色とが混ぜ合わされます。たとえば，青の線を引いたうえにXORを指定して線を重ねれば，マゼンタ(赤紫)になります。XORを使うと，画面に透明感を表現することもできます。

〔プログラム4-4〕

```
10 WIDTH40:SCREEN0,0:CLS4
20 LINE(0,0)-(100,100),PSET,1,BF
30 LINE(0,0)-(140,140),XOR,2,BF
40 LINE(0,0)-(180,180),XOR,4,BF
```

このプログラムは，光の3原色である，青，赤，緑で中が塗りつぶされた長方形を重ねて描き，XORで色がどのように重ねられるかを見てみようというものです。グラフィックスで使うカラーのうちマゼンタは青と赤，シアン（水色）は青と緑，黄色は赤と緑を組み合わせて作られることがわかります。また，3色すべてを合わせると白になることがわかります。逆に言えば，カラーコードが青（1），赤（2），緑（4）を基準に，マゼンタ（1＋2＝3），シアン（1＋4＝5），黄色（2＋4）＝6という組み合わせによって決まっていることもわかります。

LINE文はこのように多彩な機能を持っていますから，いろいろな使い方を試してみてください。

●文字表示に使えるライン文

グラフィックスでなく，通常の文字表示に対するLINE文もあります。

実際の例で理解していただくのが最もわかりやすいと思いますので，次の例を実行してみて下さい。

```
10 WIDTH 40:SCREEN0,0:CLS4
20 LINE(0,0)-(39,24),"▒",B
```

命令の書き方はグラフィックスの場合と同じですが，カッコの中は，点の座標ではなく文字表示の座標になります。80字モードでも40字モードでもそれぞれ画面いっぱいまで使うことができますから，40字モードのときは，X座標は0から39，80字モードのときには0から79となります。Yの値はどちらも，0から24までです。この値をはずれても，画面の外にはみだしてしまうだけでエラーになってしまうことはありません。その後にはどの文字を使って線を引くか決めるために，文字をダブルクォーテーションで囲んで書きます。また，グラフィックスと同様に，BやBFもあります。タイトル画面を書くときなどに便利に使えるでしょう。この場合には，グラフィックスの場合と違い，COLOR文を使って色の指定を別に行なわなければなりません。

COLOR文というのは，画面の色を設定するステートメントで，

COLOR　表示色，背景色

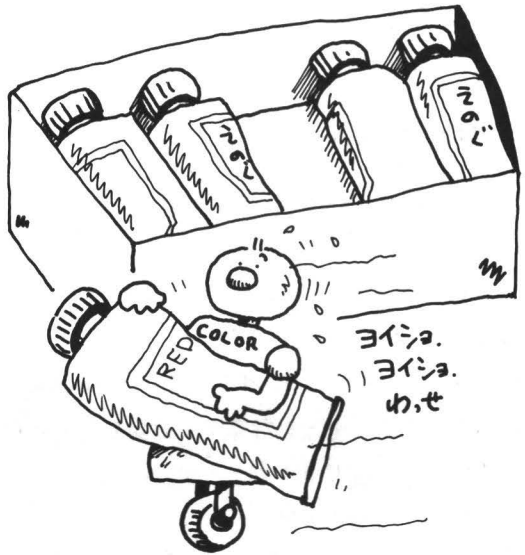
と指定し，色は色コードで指定します。X1では電源を入れたとき自動的にCOLOR 7, 0 にセットされています。ダイレクトモードで，

COLOR　1

と打ちこんでみてください。画面にOKの文字が青色で表示され，これから打ち込む文字は青色になります。

COLOR　1, 7

と入れてみましょう。画面全体が白くなって，その中に青い文字が表示されています。このように表示の色と背景の色を自由に選択できるわけです。COLOR1やCOLOR4のように省略してやると背景の色は変化しません。なお，表示色は CTRL キーと表示したい文字の色コードの数字をテンキーを使って同時に押すと，文字の色が変わります。背景の色は，CTRL キーとメインキーボードの数字を同時に押せば変わります。

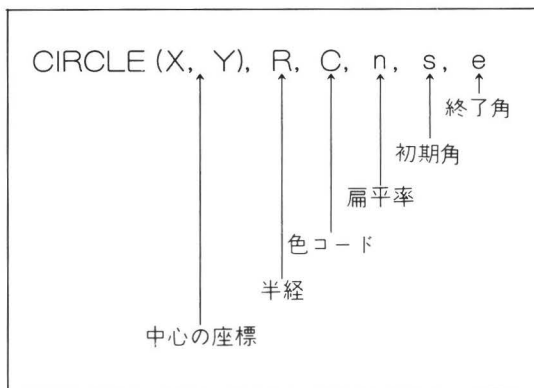
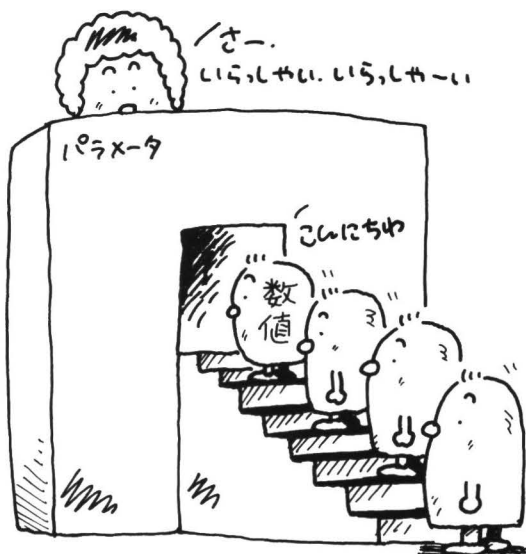
直線を引く作業はコンピュータ・グラフィックスの基本中の基本です。「何だ，こんなことか」と思わずに，いろいろと試してみてください。

4-3 わっ！ CIRCLE

●円を描く

いくら基本とはいえ線や枠ばかり描いてもおもしろくありません。今度は円を描いてみましょう。

円を描くには，円の公式を使って座標上の点を１つずつ計算し，PSET文によってドットを打っていく方法とCIRCLE文を使う方法の２とおりがあります。

CIRCLEというのは輪のことですから，CIRCLE文はその名のとおり，円と楕円を描くための専用の命令です。座標を計算する必要もなく，たった１つの命令で円や楕円を描くことができ，しかも描く速度も早くできます。

CIRCLE文の使い方は，

となります。この(X，Y)，R，C，n，s，eはパラメータと呼ばれるものです。たとえば，数字で円の面積は，S＝πr^2というふうに習いました。rが半径であることはあなたの記憶のなかに残っていますから，この式を使うことで半径の大きさがわかる円なら，どんな円の面積も計算ができます。この公式のrのように，異なった値をとることのできるものをパラメータといいます。つまり，プログラム中のパラメータは，プログラミングのときには決定しておかずに，実行時に選択の自由を残しておくもののことをいいます。上のCIRCLE文のパラメータは少々複雑にみえますが，１つ１つの意味を覚えておけば，すぐに使いこなせるようになります。

X，Ｙは描く円や楕円の中心の座標です。たとえばWIDTH 40（40字モード）で画面の中央に円の中心を置きたければ，Ｘは160，Ｙは100になります。

Ｒは円の半径です。画面内に収まる最大の半径は99になります。Ｃは色コードで，何色で描くかを決めます。Ｃを省略するとCIRCLE文が実行される前に使われていた指定色がひきつがれて使われます。LINE文やPSET文でも，色コードを省略するとそれまで使われていた色がそのままひきつがれます。

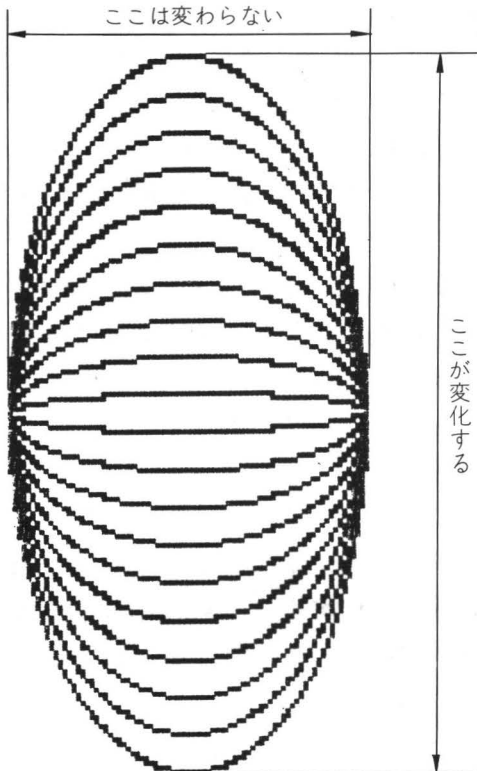
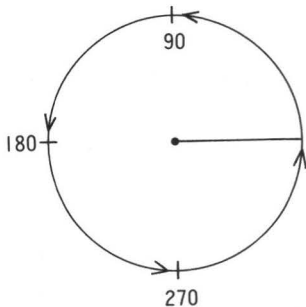
●楕円を描く

楕円を描くときには扁平率hを指定しなければなりません。正円を描くときにはhを省略します。扁平率はＹ方向の半径に対するＸ方向の半径の比率（Ｙ方向の半径÷Ｘ方向の半径）ですから楕円の横の大きさは変わらず，縦方向が大きくなったり小さくなったりすることになります。

〔プログラム４－５〕

```
10 WIDTH40 :SCREEN0,0:CLS4
20 FOR h=.1 TO2 STEP.2
30 CIRCLE(160,100),40,7,h,0,360
40 NEXT h
```

〔図４－３〕

〔画面４－２〕たまねぎの断面

ＥとＳは，円を描きはじめる角度と終わりの角度です。ＥとＳを０，360とすると円になり，途中の数値で切ってやると，弧を描くことになります。

角度は，中心から右の方向が０度で，左回り（時計の針と反対回り）に360度で円になります。省略すると０から360になります。

CIRCLE文でパラメータの省略を行なう場合，注意しなくてはいけないのは，カラーだけや扁平率だけの省略はできないという点です。どれかを省略するような場合にはそこから後もそっくり省略できるような条件でないとエラーになってしまいます。

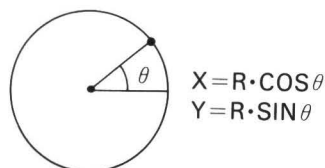
●円の公式と関数の定義

次に，CIRCLE文を使わずに，円を描く方法を見てみましょう。円の公式から円周上の点の座標を計算し，そのドットをセットすることによって円を描きます。この方法では，計算が必要なうえに同じ図形を描くにも少々時間がかかりますが，たとえば，時計の針を描こうとするときなどは，この計算を行なわなければなりません。この方法を覚えておくと，さらに複雑なトロコイド曲線などを描くときにも応用できます。

円の公式を展開すると，半径の円周上で角度θのときの座標は，$X=R\cdot\cos\theta$，$Y=R\cdot\sin\theta$で計算されます。次の例を見てください。

〔プログラム４-６〕

```
10 WIDTH40:SCREEN0,0:CLS4
20 DEFFNX(I)=COS(RAD(I))*80
30 DEFFNY(I)=SIN(RAD(I))*80
40 FOR I=0 TO 360
50 X=FNX(I)+160
60 Y=FNY(I)+100
70 PSET(X,Y,4)
80 NEXT I
```

〔図４-４〕

ここでは，FOR〜TO〜NEXT文を使って，変数θの値を０から360まで変化させながら，そのときのX座標，Y座標を計算して点の表示を行なっています。

20行と30行で，座標の計算式をDEF FNという形を使って定義しています。BASICには，sinやcosなどの関数があらかじめ用意されています。プログラム中で何度も同じ計算を行なう場合，DEF FN文を使うと，その計算式を１つの関数として使うことができるようになります。cos(X)とすると数値Xのコサインが求められるように，複雑な計算式をDEF FN文を使って１つの関数として定義しておくと，簡単に計算の結果を求めることができるというわけです。DEF FNの使い方を具体的に見てみましょう。

$$\frac{X+Y+Z}{3}$$

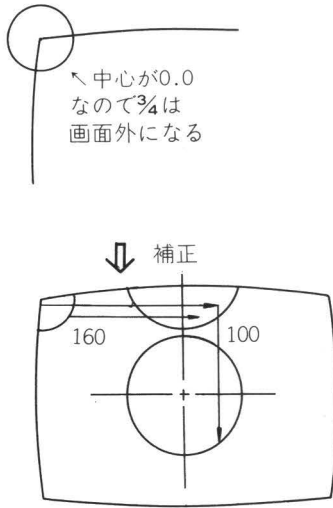
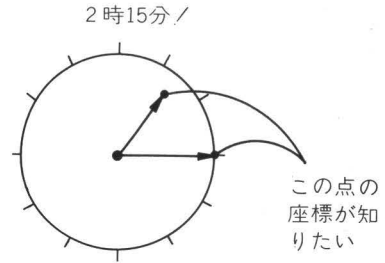
DEF FN A (X, Y, Z) = (X, Y, Z)／3

後には関数名をつけておき，カッコの中には定義する式で使われる変数をカンマで区切って入れておきます。定義した計算式が実際に使われるときには，この変数が数値に置きかわって計算されるわけです。「＝(イコール)」の右側には計算式を書きます。DEF FNで関数を定義できるのは一度だけで，定義し直すことはできません。定義しておいた関数を呼び出して使うには，FNX (n)のようにすれば，COS (n)やRAD (n)とまったく同じに使用することができます。

ここでは，FOR〜TO〜NEXTによって変化していくＩの値をラジアンに変換したものを与えて，円周上の座標を求めています。後の160と100は画面の中央に中心を移動させるための補正値です。このように補正を行なうことを，ある数値だけ数を増やしてやることから「ゲタをはかせる」ということもあります。グラフィックで画面の中央を基準としたい場合，補正値は40字モードではX＝160，Y＝100，80字モードではX＝320，Y＝100です（図４-５）。

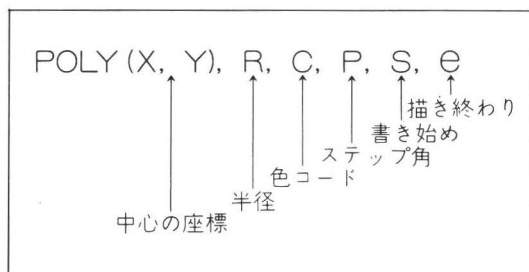
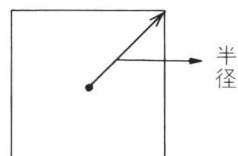
なお，〔プログラム４-６〕をWIDTH 80に直して描くと縦に長い楕円が描かれることになってしまいます。これはドットの並び方の縦と横の比率が変わってしまうためで，円を描くためには，X方向の半径を２倍にしておけばうまくいきます。20行の「＊80」を「＊160」にすればよ

いわけです。

〔図 4 − 5〕

〔図 4 − 6〕

　このように計算式によって座標を求める方法を使うと，どんな複雑な曲線でもその曲線の方程式がわかっていれば描くことができます。ユーザー定義関数の書きかえだけでそのまま利用できますので，例のプログラムを頭に入れておくと役に立ちます。ただし，CIRCLE文で描かれた円と違い，半径が大きいと点と点の間が離れてしまいます。後で説明するPAINT文で円を塗りつぶしたりする場合には不都合が生じますので，うまく使いわけることが必要です。計算式を使う方法は，たとえば時計の針のように円の中心から円周上の１点に線を引くときなどに必要となります。CIRCLE文を使って円を描くと左回りに描かれましたが，この計算式による方法を使うと右回りに描かれます。

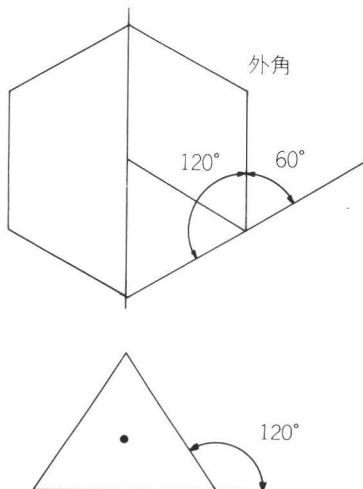
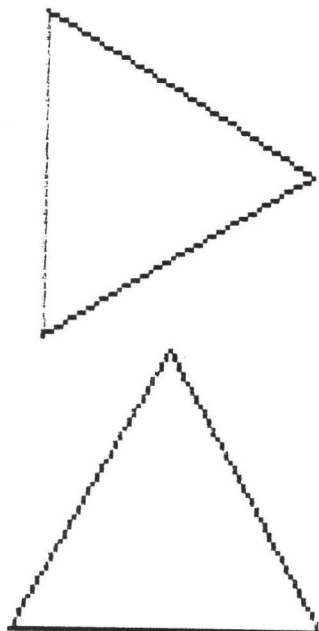
4-4 ひとで？ 星？

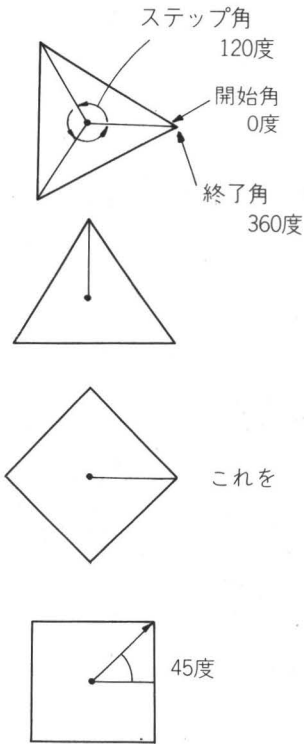
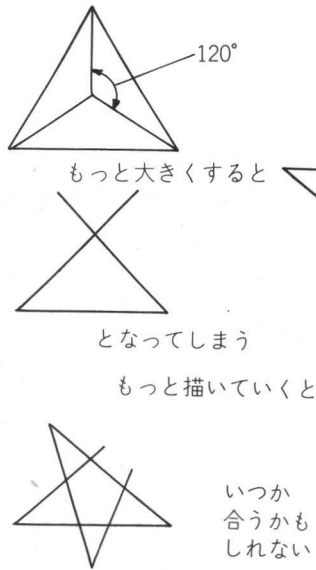
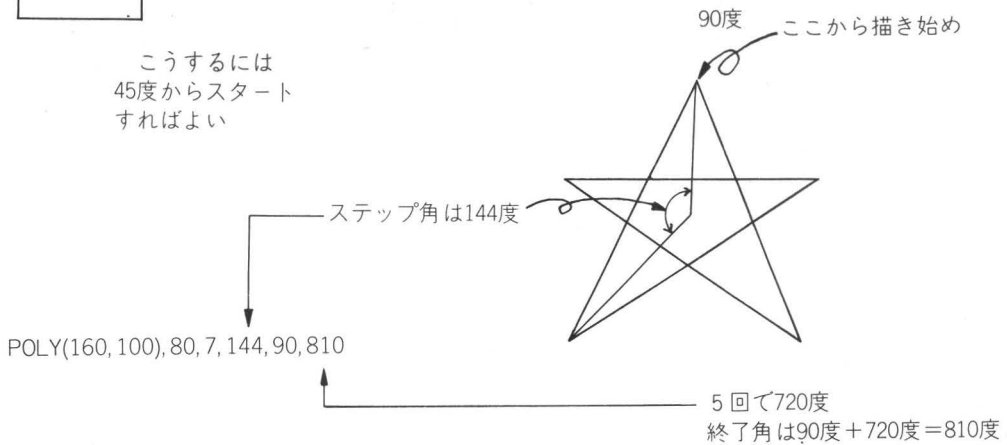
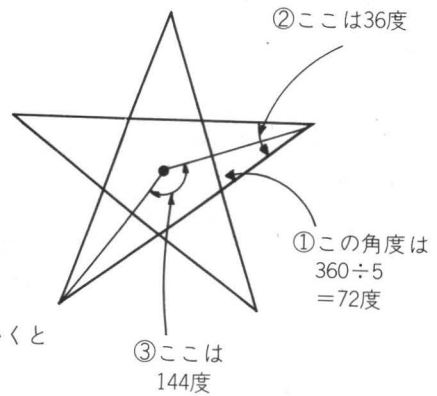
●POLYでひとで？

多角形を描く命令がPOLYです。これまで長方形や正方形を描くLINE文と円を描くCIRCLE文を見てきました。それ以外の多角形を描くためにPOLY文が用意されています。POLY文の書き方は，

〔図4－7〕

です。

CIRCLE文と同じように中心の座標，半径，色コードの順に指定します。半径というのは，中心から頂点までの距離です（図4－7）。Pはステップ角で，言いかえればその多角形の1つの外角と考えればよいですから，n角形なら$\frac{360}{n}$度と指定すればよいわけです(図4－8)。Sとe はCIRCLEと同じく，描きはじめと終わりを指定します。各パラメータの省略の仕方もCIRCLE文と同じですが，ステップ角Pを省略すると1が入り，円が描かれます。

まず，正三角形を描いてみましょう。

〔図4－8〕

〔画面4－3〕

```
10 SCREEN0,0:CLS4
20 POLY(160,50),40,4,120,0,360
30 POLY(160,150),40,6,120,90,450
```

中心の座標を（160，50）とします。半径，つまり頂点までの距離を40としましょう。ステップ角は360÷３ですから，120になります。色コードは白ですから７，そして０度から360度までとして実行すると，画面４－３のように頂点が右を向いた正三角形になります。これを，頂点を上に向けるためには，開始角を90度にします。終了角はここから360度の位置ですから450度となります。今度はうまく上を向いた正三角形が描けました。

正四角形（正方形）ではどうでしょうか。開始角を０度とするとやはり斜めに描かれてしまいます。最初の頂点の位置を45度動かして開始角を45度，終了角は360度を足した405度と指定すると，正方形が描かれます（図４－９）。

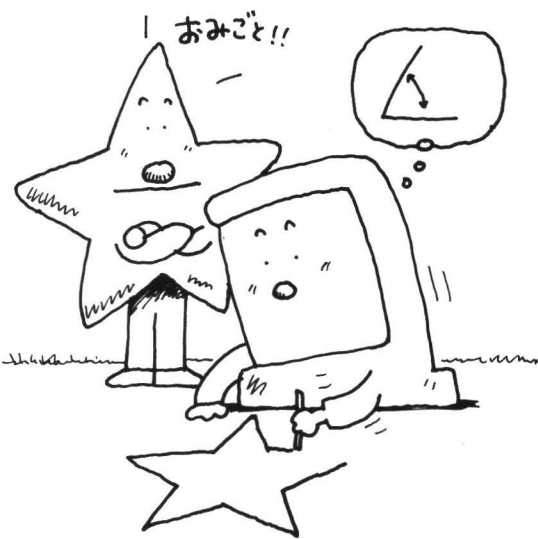
開始角と終了角の差が360度よりも大きくなる場合もあります。それは，ステップ角が120度以上の場合です。ステップ角120度で正三角形が描かれるわけですから，これよりステップ角を大きくすると，正多角形は描けなくなります（図４-10)。終了角を増やして描き続けると線の描きはじめの点と描き終わりが少しずつ近づいてきます。一筆書きの星を思い出してみると，５回折り曲げたところで星が描き終わります。図４-11によって考えてみると，星形のステップ角は144度になっています。そして折り曲げ回数が５回ですから，開始角から144度×５＝720度動かせばよさそうです。

これでうまく星が描けたはずです。半径や色を変えて，いろいろ試してみてください。

なお，ステップ角を０度とすると，中心から開始角の頂点の座標までの直線，つまり半径を描きます。

最後に多角形のステップ角と描きはじめと終わりを一覧表にしましたので，参考にしてください。

表４－２

	ステップ角	始め～終わり
三角形	120度	0 ～360度 90～450度
四角形	90度	45～405度
五角形	72度	90～450度
六角形	60度	0 ～360度 30～390度
星形	144度	90～810度

4-5 色，いろいろ ありまして！

●色，いろいろ(1)

これまで，何げなくグラフィックスの命令の中で色を使ってきました。ここで一度きちんと覚えておきましょう。これまで一般的な呼び方にならって「色コード」と呼んできましたが，X１では正確には「パレット（PALET）コード」と呼びます。色コードは表のように決められていて変化することはありません。

パレットというのは，絵具を混ぜ合わせたり薄めたりするのに使う，あのパレットと同じです。今まで使ってきたX１のパレットには，色コードと同じ絵具がそっくり入っていました。X１のパレットにはその区切りごとに番号がついており，これをパレットコードと呼びます。これまでは色コードとパレットコードが一致していました。ときによっては，このパレットにのせる色を変えてしまうこともできます。たとえば，

PALET 2, 4

とすると，パレットの２番目のところに緑（色コードでは４）がセットされます。こうなっていると，たとえば，

PSET (X, Y, 2)

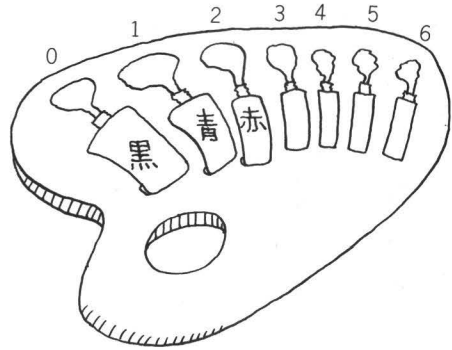
としても，赤ではなく緑のドットが画面に描かれます。。X１のグラフィックスでは，LINE文にXOR

そして，PALET文でセットする色を指定し直すと，パレットの中だけでなく，すでに描かれた部分の色も変化してしまいます。これをうまく利用すると，いろいろおもしろい効果が得られます。

アプリケーションテープの中は，コマが回転するように見えるプログラムがありました。PALET文で色を変化させてコマが回転するように見せているのです。

次のプログラムはこれと似た効果をねらったものです。

〔プログラム４-７〕

```
10 WIDTH40:SCREEN0,0:INIT:CLS4
20 FOR I=0 TO 100
30 CIRCLE(I,100),I,(I/10)MOD7+1,1,0,360
40 NEXT ,
50 FOR I=1 TO 100
60 FOR J=1 TO 7
70 PALET(I+J) MOD 7+1,J
80 NEXT:NEXT
```

CIRCLE文で円を10個描くごとに色を変えます。そして，50行からの部分でパレットを切りかえておもしろい効果を見せています。

この中の2カ所で使われているMODというのは，割り算をしたときの余りを求める演算子です。演算子というのは，+，−，*，／などのような計算の内容を表現する記号です。

30行では，MODによって円を10回描くごとに色（パレットコード）を変える操作を行なっています。Ｉの値をいろいろ変えて試してみると，この部分の働きがわかると思います。70行では，パレットにのせる色の順番はそのままにしておいて，１つずつパレットの番号をずらしています（図4-12)。これによって流れるような色の変化を表現しているのです。

〔図4-12〕

10行のINITは，画面の初期化のために入れてあります。これで，前に入っていたプログラムの影響をうけることなく新しいプログラムが実行されるわけです。

パレット文によって変化するのは，LINE，CIRCLE，PSETなどのグラフィック命令で描かれた線や点だけで，キーボードから打ち込む文字や記号（テキストといいます）にはなんの影響も与えません。

●色・いろいろ(2)

Ｘ１では，黒を含めて８色の色を使うことができます。グラフィックスのいろいろな命令を学びながらこれら８色の色を使ってみました。ここで，もう１つ新しいグラフィックスの処理を見てみましょう。

たとえば，青で中を塗りつぶした四角形を描くには，

```
LINE (0, 0) — (80, 80), PSET, 1, BF
```

とすればよかったわけですが，これを四角形の枠の色とその内側の色とを別々の色で塗り分けることもできます。

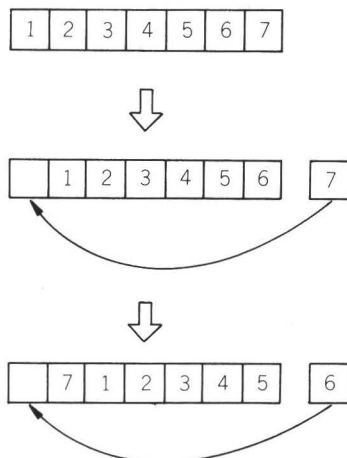
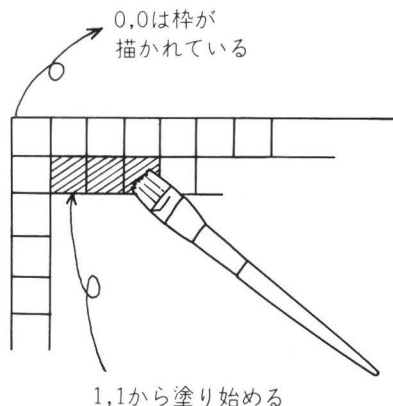
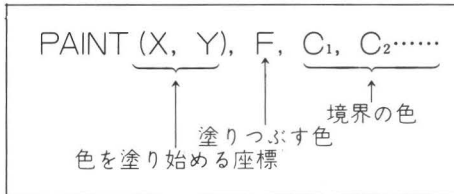
```
CLS 4
```

として画面を消してから，まず枠を描きます。

```
LINE (0, 0) — (80, 80), PSET, 7, B
```

で白枠が描かれます。この中を，赤で塗るようにすると，

```
PAINT (1, 1), 2, 7
```

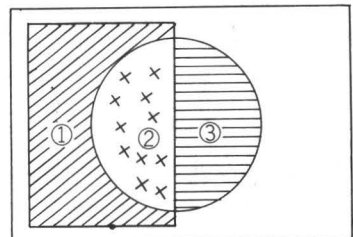
とします。PAINT文は，線によって閉じた図形の中を塗りつぶす命令（図4-13）で

〔図4-13〕

と書きます。(X, Y)は色を塗りはじめる座標です。この例では，(X, Y)を(1, 1)としています。座標(0, 0)には枠が描かれているため，その1つ内側の点を指定しているというわけです。Fは塗りつぶす色，C_1，C_2……は境界の色です。

ここでの例を見てみると，色コード2の色（赤）で色コード7の色（白）の境界まで塗りつぶします。境界の色は，PAINT文で塗ろうとする色と境界色が同じなら省略してもかまいません。また，境界色が複数の場合は，その数だけ書いておかなければなりません。

〔プログラム4－8〕の場合，まず四角形を描き，ちょうど半分重なった位置で円を描きます。そして40行で，図4-14の①の部分を青く塗りつぶします。四角形の枠は青ですから，枠内に入りこんでいる円の白を境界色として置いています。この①の部分の境界色は，青と白の2つの色があるわけですが，青の方は塗る色と同じですから省略しています。40行のPAINT文で指定しているのは，塗りつぶす青の色コードと，境界色の色コードの2つだけなのです。②の部分も同じです。③の部分は，塗ろうとする色がマゼンタなのに対して境界色は青と白の2色ですから，この両方を境界色として指定しなければなりません。

〔図4-14〕

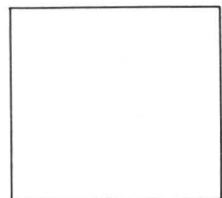
〔プログラム4－8〕

```
10 WIDTH40:SCREEN0,0:INIT:CLS4
20 LINE(0,0)-(160,199),PSET,1,B
30 CIRCLE(160,100),80,7,1
40 PAINT(1,1),1,7
50 PAINT(161,100),2,1,7
60 PAINT(159,100),3,7,2
```

塗りはじめの点の位置の指定は，LINE文で枠を描いた場合は描きはじめのX座標，Y座標の値に，それぞれ1を加えた座標点とし，CIRCLEやPOLYで円や多角形を描いたときは中心にすると間違いがありません（図4-15)。

〔図4-15〕

PAINT文で色の塗りつぶしを行なう場合，つぶそうとする枠がわずか1ドットでも欠けていると，その部分から外ににじみ出してしまいます。

〔プログラム4－9〕

```
10 WIDTH40:SCREEN0,0:INIT:CLS4
20 LINE(0,0)-(80,80),PSET,1,B
30 LINE(40,40)-(100,40),PSET,2
40 PAINT(1,1),3,1
```

この場合，青で描いた枠を30行で描いた線が横切っているため，その部分が途切れてしまってそこから色がにじみ出してしまいます。こうなった場合，SHIFT＋BREAKで止めない限り画面いっぱいを塗りつぶすまでプログラムは止まり

ません。

PAINT文で黒（0）を使う場合は，その部分がすでに0以外の色で塗られていなくてはなりません（プログラム4-10）。

〔プログラム4-10〕

```
10 WIDTH40:SCREEN0,0:INIT:CLS4
20 LINE(0,0)-(80,80),PSET,1,B
30 PAINT(1,1),5,1
40 LINE(20,20)-(60,60),PSET,0,B
50 PAINT(21,21),0
```

また，一度色コード0以外の色で描いた枠をもう一度色コード0で描き直しておけば，色コード0を境界色として指定することもできます（プログラム4-11）。

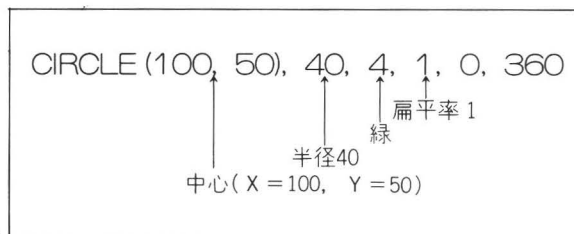
〔プログラム4-11〕

```
10 WIDTH40:SCREEN0,0:INIT:CLS4
20 LINE(0,0)-(80,80),PSET,1,B
25 PAINT(1,1),2,1
30 LINE(0,0)-(80,80),PSET,0,B
40 PAINT(1,1),3,0
```

ただし，枠だけを色コード0で描き直してもだめで，色コード0の枠の内側には必ず黒以外の色が塗られていなければなりません。

●エラーが出たぞ！（Illegal function call）

CIRCLEやPOLYなどの命令には，パラメータがたくさんついており，覚えるまでが大変そうです。これらのパラメータには，使用範囲に限界のあるものがあります。その範囲外の数値を指定すると，「Illegal function call」というエラーメッセージがでます。パラメータの多い命令の場合，並べる順序を間違えたり，途中で抜かしたりすることでも，このメッセージにお目にかかることになります。たとえば，CIRCLE文で，

とするところを，

CIRCLE (100, 50), 4, ,40 ……

とするとこのメッセージが表われます。この場合，半径は4でもかまわないのですが，色指定のところで40がエラーとなっているわけです。

この他にもたとえば，LOCATE文のエラーもあります。WIDTH 40のとき，画面の一番下の行の一番右側の端に表示させようとして，LOCATE 40, 25と指定してもエラーがでます。0行目と0文字目が含まれているので，実際にはこの座標は39, 24なのです。

また，CONSOLE文でスクロール範囲を指定している場合，LOCATE文を使ってスクロール範囲外に表示させようとして，エラーを起こすこともしばしばあるようです。

CIRCLE文などのパラメータの多い命令のパラメータは重要なものから順に並んでいますから，

よく注意して使えば，このエラーを起こすこともなくなります。「習うより慣れろ」ですね。

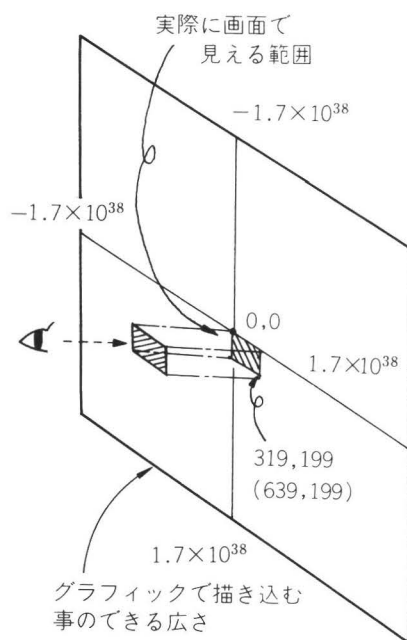
4-6 X1のグラフィックス，こんなこともできる

●窓を開ける

「いっしょうけんめい描いた図形をもう少し大きくしたい」，あるいは「複雑な図形の一部分を拡大して詳しく見たい」——このようなときにX1では，WINDOWという命令を使います。グラフィックス画面の表示の範囲を変化させ，希望の図形が得られます。

X1のグラフィックス座標は，X方向が0から319（80字モードの場合は639），Y方向が0から199までであることは既に説明しました。この座標を絶対座標と呼びます。ディスプレイテレビの画面のサイズに収まっている範囲の座標のことです。

これに対し，グラフィックスの命令によっていろいろなものを描くことができる範囲は，±1.7×10^{38}四方の広大な広さを持っています。こちらの座標を論理座標と呼びます。普通は，この絶対座標と論理座標の座標位置は一致しています（図4-16）。

〔図4-16〕

絶対座標で表わされる画面は，この広大な論理座標を見るための「窓」と考えることができます。

「窓」を移動させると論理座標上の希望の部分をのぞくことができます。しかも，この窓の中に映る範囲を変化させることもできます。

これまでLINEやCIRCLEなどのグラフィックス関係の命令で指定していたのは，実はすべてこの論理座標上の座標点であったのです。次の例のプログラムを走らせてみましょう。

```
10 WIDTH40:SCREEN0,0:INIT:CLS4
20 POLY(319,199),200,6,144,90,810
```

POLY命令で描かれる星形の中心は，ディスプレイテレビの画面（絶対座標）では右下のすみになります。当然のことながらディスプレイテレビの画面に表示されるのは星形の左上の４分の１の部分だけになります。そこでWINDOW文によって，絶対座標内，つまりディスプレイテレビの画面内に表示される論理座標の範囲を広げてやります（図４-17)。

```
15 WINDOW(0,0)-(160,100),(0,0)-(639,399)
```

「窓」に映る範囲が変化しますから星形の全体を見ることができるようになります。絶対座標の４倍の広さの範囲に描かれた星形の全体を表示させているわけです（図４-18)。

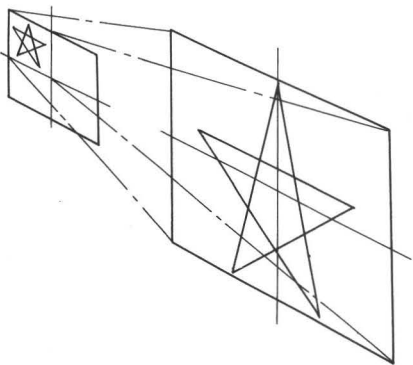
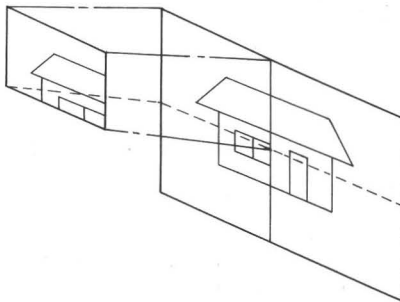
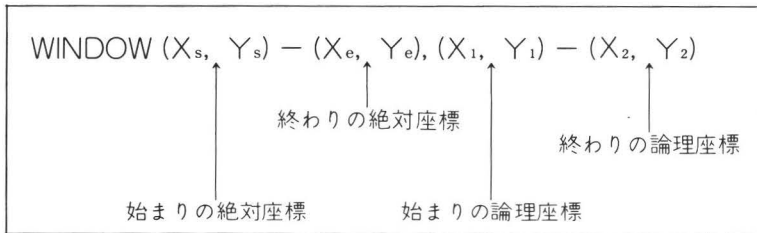
〔図４-17〕

〔図４-18〕

さらに，論理座標を映す「窓」の大きさを，絶対座標全体（ディスプレイテレビの画面）ではなく，その一部分にすることもできます。15行を次のように変えてみましょう。

```
15 WINDOW(0,0)-(319,199),(0,0)-(639,399)
```

「窓」を小さくしたので，画面左上の部分に星形が描かれます。この場合は，縦横それぞれ４分の１，面積で16分の１になったことになります（図４-19)。

〔図４-19〕

〔図４-20〕

上記が一般的な書き方です。X_s, Y_s, X_e, Y_eは絶対座標で，画面モードによる最大値を越えたり（$X_e>319$ (639)，$Y_e>199$），負の値を使うとエラーになります。また，WINDOW文によって切りかえを行なっても，既に描かれていたものがそのまま拡大，縮小されるわけではなく，再度書き込みが行なわれたときに，指定された状態で表示されることになります。

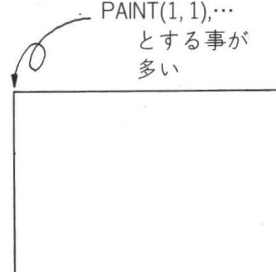
次の〔プログラム4-12〕は家の形を描き，希望によって拡大，縮小を行なうものです。

〔プログラム4-12〕

```
10 WIDTH40:SCREEN0,0:CLS4
20 WINDOW:GOSUB "HOUSE":GOSUB "FLAME"
30 LOCATE0,23:COLOR7:PRINT"ｶｸﾀﾞｲ --- 1   ｼｭｸｼｮｳ --- 2 ";
40 REPEAT:Z$=INKEY$(1):UNTIL Z$="1" OR Z$="2"
50 IF Z$="1" GOSUB"*2" ELSE GOSUB"/2"
60 SCREEN0,0:GOTO30
70 LABEL"*2"
80 LOCATE0,23:PRINTCHR$(5):LOCATE0,23:PRINT"SELEST 1 - 4 ";
90 REPEAT:A$=INKEY$(1):UNTIL Z$>"0" AND Z$<"5"
100 SCREEN1,1:CLS4:ON VAL(A$) GOSUB 140,150,160,170:GOSUB "HOUSE
"
110 LOCATE0,20:PRINT"Hit Any KEY"
120 REPEAT:X$=INKEY$:UNTIL X$<>""
130 RETURN
140 WINDOW(0,0)-(319,199),(0,0)-(160,100):RETURN
150 WINDOW(0,0)-(319,199),(160,0)-(319,100):RETURN
160 WINDOW(0,0)-(319,199),(0,100)-(160,199):RETURN
170 WINDOW(0,0)-(319,199),(160,100)-(319,199):RETURN
180 LABEL"/2"
190 LOCATE0,23:PRINTCHR$(5):LOCATE0,23:PRINT"ﾊﾌﾞﾝ ﾉ ｲﾁ ? 2 - 6 "
;
200 REPEAT:A$=INKEY$(1):UNTIL VAL(A$)>1 AND VAL(A$)<7
210 N=VAL(A$)
220 SCREEN1,1:CLS4:WINDOW(0,0)-(319/N,199/N),(0,0)-(319,199):GOS
UB "HOUSE"
230 LOCATE0,20:PRINT"Hit Any KEY"
240 REPEAT:X$=INKEY$:UNTIL X$<>""
250 RETURN
260 END
270 LABEL"HOUSE"
280 LINE(100, 90)-(220,170),PSET,4,BF
290 COLOR2:LINE(120,40)-(200,40)-(240, 90)-(80, 90)-(120,40)
300 PAINT(125,45),HEXCHR$("00FF1100FF44"),2
310 PAINT(195,45),HEXCHR$("00FF1100FF44"),2
320 LINE(120,110)-(170,140),PSET,7,B
330 LINE(121,111)-(169,139),PSET,0,BF
340 LINE(120,125)-(170,125),PSET,7
350 LINE(145,110)-(145,140),PSET,7
360 LINE(180,110)-(210,170),PSET,1,BF
370 LINE(130,20)-(150,60),PSET,6,BF:COLOR7
380 RETURN
390 LABEL"FLAME"
400 LINE(0,0)-(319,176),PSET,5,B
410 LINE(160,0)-(160,176),PSET,5:LINE(0,100)-(319,100),PSET,5
420 LOCATE0,1:PRINT"1":LOCATE20,1:PRINT"2":LOCATE0,13:PRINT"3":L
```

```
OCATE20,13:PRINT"4"
430 RETURN
```

家の形は，サブルーチン"HOUSE"で描いており，絶対座標の中央に描けるよう，パラメータが決めてあります。

この画面を縦横４つの部分に割り，拡大しようと希望する部分を絶対座標いっぱいに表示します。この指定を行なうのが，140～170行の部分です（ここでは，論理座標のどの部分で表示するかを指定しています)。実際の画面の状態と見比べて，どのように指定しているか考えてください。

縮小を行なうサブルーチンでは，今度は絶対座標の表示範囲の方を変化させています。このような縮小を行なった場合，ドットの間隔が圧縮されるため，PAINT文がうまく働かない場合があります。通常，PAINT文の塗りはじめの位置は，図４-21のような決め方をすることが多いのですが，WINDOW文を使って縮小して表示するような場合，塗りはじめの位置を枠内の中心に置く方がよいでしょう。

〔図４-21〕

また，拡大を行なう場合，塗りはじめの位置が絶対座標外，つまり画面の外にあるとペイントされなくなりますから，300行，310行のように２カ所指定しておくとよいでしょう。

このプログラムでは，次に研究する「時計」のプログラムと同じように，スクリーンの０と１を使いわけ，０の方に元の絵を，１の方に拡大，縮小されたものを描くようにしてあります。また，キー入力には，INPUT文を使わず，すべてINKEY$によってワンキー入力になるようにしてありますから，この使い方もよく覚えておいてください。

4-7 パソコン画面に時計をかこう！

●時計を動かしてみよう

ここまでで，グラフィックの線を引いたり，円を描いたりする命令がだいたいわかりました。みなさんもいろいろな形や図を描くことができるようになったことと思います。しかし，いつまでも線を描いているだけでは進歩がありません。そろそろ動画に挑戦してみたいところです。

応用として時針・分針・秒針のついたアナログ時計を作ってみましょう。

まず，針を動かす方法を考えます。この時計の基礎となる時刻は，もちろんX１の内蔵時計を利用します。まず，TIME$から秒を取り出します。文字列処理関数のうちRIGHT$を使えば，TIME$の右から２文字，つまり秒が取り出せます。この他に，分や時も必要ですが，MID$，LEFT$という文字列処理関数を使えばよいですね。秒針を描く部分をサブルーチンとして使うように，あらかじめラベルをつけておきます。

```
600 LABEL "HARI秒"
610 BO=VAL(RIGHT$(TIME$,2))
```

VALは数値文字列を数値に変換する関数です。文字列として取り出したTIME$の秒の部を数値に直しておき，後で計算を加えられるように準備しておきます。

秒針は60秒で１回転，つまり360度回りますから，１秒では，$\frac{360}{60}$で６度だけ動きます。この６度だけ動いた針の位置を計算するには，円の公式を使います。

この公式では，中心から右にいった位置を０として右回りになっていましたから，０秒の位置はこのままでは時計の３時の位置になってしまいます。これを真上の12時の位置を０にするには，左回りに90度戻してやればよいわけです。この２つの点を考えると，秒針の角度は，BO＊６-90ということになります。変数BOは現在の時刻の秒です。さらにこれをラジアン単位に変換しなければなりませんから，プログラム中では，

```
620 BP=RAD(BO*6-90)
```

としておきます。

針の長さは，時計が丸い形をしていますからその半径になります。時計の半径をRBとし，中心の座標を，X＝160，Y＝88とすると，円の公式から針の先の座標を計算することができます。

```
630 BX=COS(BP)*RB+160:
    BY=SIN(BP)*RB+88
```

変数BX，BYの値が針の先端のX座標，Y座標です。中心からこの座標に向かって直線を引けば，秒針が描けることになります。

```
640 LINE(165, 88)-(BX, BY), PSET, 6
660 GOTO 600
```

実際に動かしてみましょう。ここでは画面の設定も何もしていませんから，ダイレクトモードで，

```
SCREEN 0, 0;CLS4:WIDTH 40
```

とします。時計の半径RBの値も決めてやらなければなりません。これを48とします。それではプログラムを走らせてみましょう。GOTO文によって動かします。LIST文やRUNのように，GOTO文もダイレクトで使用することができます。

RB＝48：GOTO 600

GOTO文を使わずRUNを命令してしまうと，せっかくこの前に指定した変数RBの値が消えてしまいます。

確かに１秒ごとに針が動いていますが，前の針を消していませんから放射状の線が描かれてしまいます。しかし，１秒ずつ針が動いていく様子はわかると思います。うまく動いたら，サブルーチンとして使うために，660行を，

660 RETURN

と直しておきます（プログラム４-15）。

時，分の針もまったく同じように作ることができます。まず，針を動かすメインルーチンだけを作っておきます。秒・針の時計がうまく動けば，時針，分針を同じように書き加えてやればよいということになります。

〔プログラム４-15〕

```
600 LABEL "HARI*"
610 BO=VAL(RIGHT$(TIME$,2))
620 BP=RAD(BO*6-90)
630 BX=160+COS(BP)*RB:BY=88+SIN(BP)*RB
640 LINE(160,88)-(BX,BY),PSET,6
660 RETURN
```

●針を消す

針を時計らしく動かすには，新しい針を描いたら前の針を消してしまわなければなりません。次の〔プログラム４-13〕を見てください。

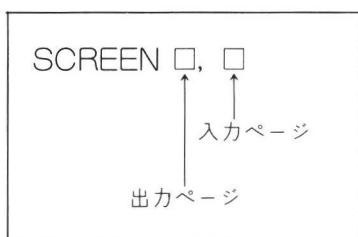
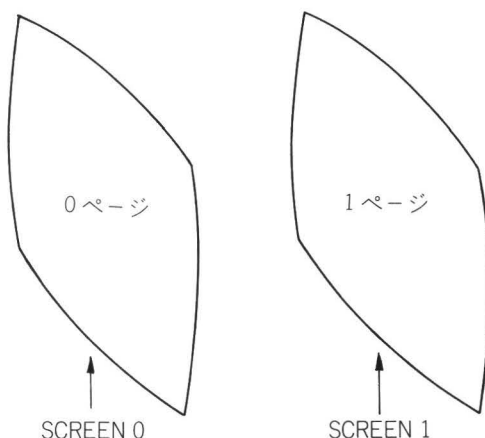
〔プログラム４-13〕

```
100 WIDTH 40:CLS 4:RB=48
160 SCREEN 0,0
170 LABEL "MAIN"
180 GOSUB "HARIREST"
190 GOSUB "HARI*"
230 GOTO 170
500 LABEL "HARIREST"
520 LINE(160,88)-(BXR,BYR),PRESET,6
550 RETURN
600 LABEL "HARI*"
610 BO=VAL(RIGHT$(TIME$,2))
620 BP=RAD(BO*6-90)
630 BX=160+COS(BP)*RB:BY=88+SIN(BP)*RB:BXR=160+COS(BP-RAD(6))*RB
:BYR=88+SIN(BP-RAD(6))*RB
640 LINE(160,88)-(BX,BY),PSET,6
660 RETURN
```

針は，LINE文で描きましたから，これを消すための命令は，LINE文のPRESETでよいはずです。新しい針を描いたときに消す針は１秒前の位置に描かれていたものですから，角度にして６度戻ったところになります。新しい針の座標を計算するときは，この角度に対する座標を一緒に計算しておくようにします（630行）。そして，PRESET（点を消す）モードでLINEを描くことによって，１秒前の針を消します（520行）。

次に，画面の初期設定をする部分を作ります。初期設定というのは，これから組もうとするプログラムのはじめに指定してやるもので，どの画面を使うか，文字のモードをどうするかなどを決める作業です。このプログラムでいうと，RBは，前にもありましたが秒針の長さです。後で，この部分に分針や時針の長さもつけ加えることになります。

160行のSCREEN命令は，今までに何回かプログラム中にでてきていますが，あまり詳しく説明していなかったので，ここでちょっと詳しく見てみましょう。X１は40字モードのときグラフィック画面を２つ別々に独立して持つことができますが，SCREEN　0，0というのはこのうちの「１ページ目を使うぞ」と指定しているものです。

SCREEN文に続けてすぐディスプレイテレビの画面に出すページ，つまり目に見えるページ（出力ページ）を指定し，次にLINE文やPSETなどのグラフィック命令や文字の書き込みを行なうページ（入力ページ）の順に指定します。

〔図４-22〕

２枚の画面のうち最初のページを０ページ，２枚目を１ページとしています。ですからSCREEN　0，1と指定するとどうなるでしょうか。ちょっとやってみましょう。

画面に何かが表示されている状態でダイレクトで，

SCREEN　0，1（SC．0，1）

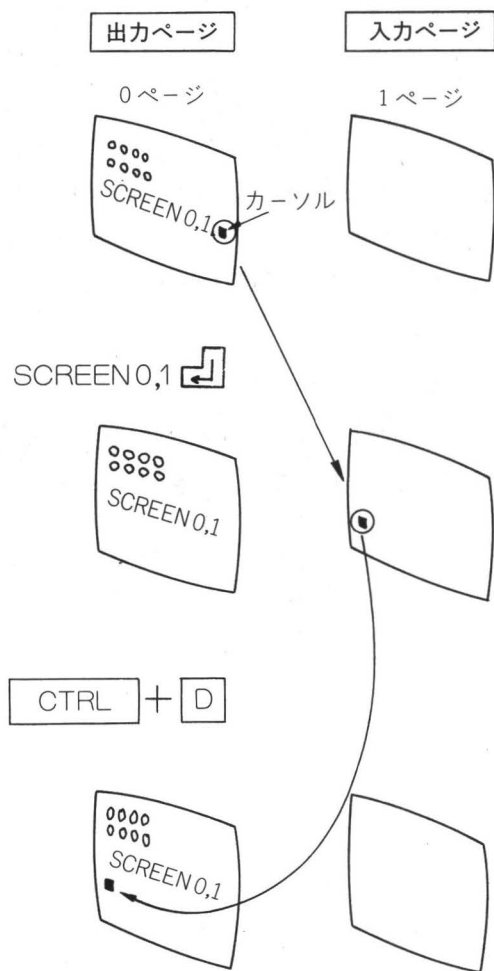
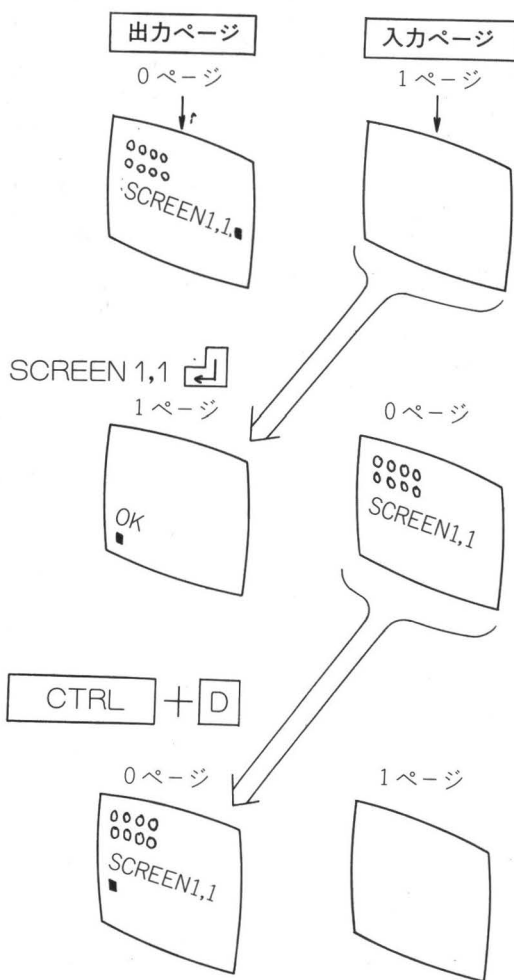
としてみます。カーソルが見えなくなってしまいました。これは，０ページが見えていて，１ページに書き込む指定ですから，文字の表示される位置を示すカーソルは２ページ目の方にいってしまい，画面に表示されている１ページ目からは消えてしまったように見えるわけです。元に戻すには，［CTRL］キーと［D］を一緒に押します。［CTRL］＋［D］は，画面の設定を初期の状態に戻す働きを持っています。

もし画面の状態がおかしくなったり，カーソルが消えてしまったときには［CTRL］＋［D］と覚えておいてください。

今度は，

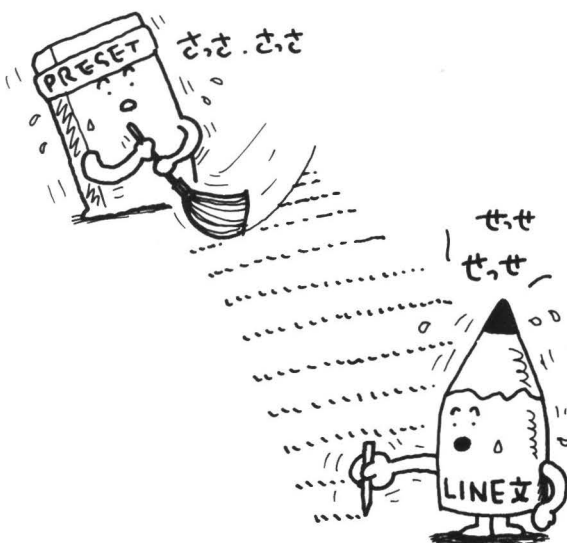
SCREEN　1，1

としてみます。一瞬の間に画面が切りかわり，1ページが表示されます。今度はカーソルがでていますから，0ページとは別のものを書くことができます。2つの画面を切りかえていろいろと試してみましょう。

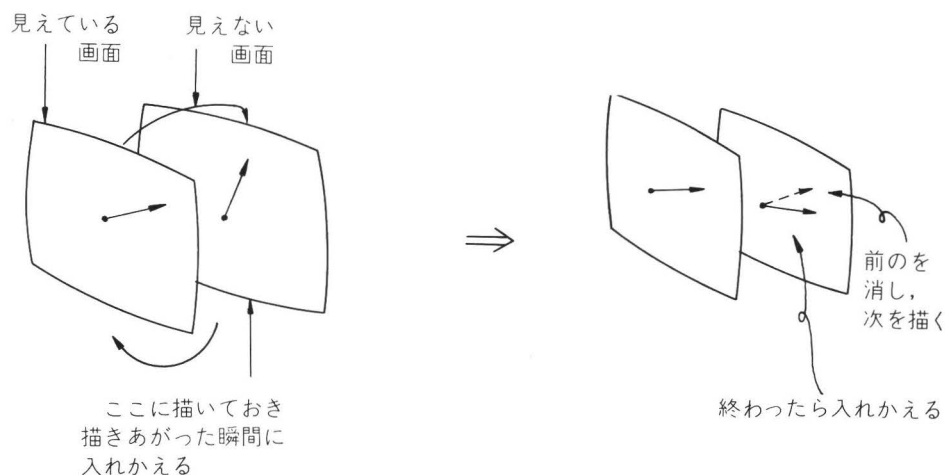
SCREEN文はちょっとはじめはわかりにくいと思いますが，よく使われる応用範囲の広い命令ですのでぜひマスターしましょう。

CLS 4はグラフィック画面1，2，3，とテキスト（文字）画面を同時にクリア（きれいに消す）します。CLSだけで何もつけないときには，テキスト（文字）画面だけをクリアし，CLS 1, CLS 2, CLS 3は，それぞれグラフィック画面の1，2，3をクリアすることになります。

170行から230行のGOSUB文の集まりがプログラムの中心となっている部分です。500行から550行までに，針を消す部分を1つのサブルーチンとしてまとめてあります。メインルーチンでは，まず針を消すサブルーチンを呼び，次に新しい針を描くサブルーチンを呼び出しています。これを繰り返すことによって秒針を動かしているのです。

うまく時計が動くかどうか，プログラムに誤りがないことを確かめて，RUNしてみましょう。

このプログラムのように，新しい位置に描いたら前の位置の表示を消す，というのが画面上で絵表示を動かすテクニックのもっとも基本となるものです。しっかりと理解してください。

●マルチスクリーンの利用

どうやら秒針をそれらしく動かすことができました。しかし，よく見ると針の根もとがちらついてどうも目ざわりです。ここでは少々高度なテクニックを使ってちらつきを直してみましょう。

〔図4-25〕

ちらつきの原因は何かと考えてみると，針を描いているLINE文と消しているLINE文とが針の根もとで重なっているためであることがわかります。

そこでそれを活用するために，2枚のグラフィック画面を活用してみましょう。SCREEN命令では，実際に目に見えるページと，書き込みを行なうページを別に指定することができることは，もうみなさんおわかりですね。これを利用して，まずディスプレイテレビの画面では見えないページで針を描いたり，前に描いた針を消したりといった作業をします。そして，描き終わった瞬間にこの2つのページを入れかえます。入れかえるといっても，ほんの一瞬の作業です。再び画面には見えないページで仕事をし，終わったらまた入れかえます。こうすると，実際にディスプレイテレビで見えるページのうえでは，描いたり消したり，といった作業は一切行なわれませんから，ちらつきのないきれいな線が得られます。画面上で図形などを動かすときにこの方法（マルチスクリーン）を使うと，自然な動きを得ることができます。

それでは〔プログラム4-14〕を考えてみましょう。

〔プログラム4-14〕

```
100 WIDTH 40:CLS 4:RB=48:RF=44:RJ=40
160 SC=0:CS=1
170 LABEL "MAIN":SCREEN SC,CS
180 GOSUB "HARIREST"
190 GOSUB "HARI秒"
200 GOSUB "HARI分"
210 GOSUB "HARI時"
220 SWAP SC,CS
230 GOTO 170
500 LABEL "HARIREST"
520 LINE(160,88)-(BXR,BYR),PRESET,6
530 LINE(160,88)-(FXR,FYR),PRESET,5
540 LINE(160,88)-(JXR,JYR),PRESET,3
550 RETURN
600 LABEL "HARI秒"
610 BO=VAL(RIGHT$(TIME$,2)):IF BO=BC GOTO 610
620 BP=RAD(BO*6-90)
630 BX=160+COS(BP)*RB:BY=88+SIN(BP)*RB:BXR=160+COS(BP-RAD(6))*RB
:BYR=88+SIN(BP-RAD(6))*RB
640 LINE(160,88)-(BX,BY),PSET,6
650 BC=BO
660 RETURN
700 LABEL "HARI分"
710 FU=VAL(MID$(TIME$,4,2))
720 FP=RAD(FU*6-90)
730 FX=160+COS(FP)*RF:FY=88+SIN(FP)*RF
740 FXR=160+COS(FP-RAD(6))*RF:FYR=88+SIN(FP-RAD(6))*RF
750 LINE(160,88)-(FX,FY),PSET,5
770 RETURN
800 LABEL "HARI時"
810 JI=VAL(LEFT$(TIME$,2))
820 JP=RAD((JI*30+FU*.5)-90)
830 JX=160+COS(JP)*RJ:JY=88+SIN(JP)*RJ
840 JXR=160+COS(JP-RAD(.5))*RJ:JYR=88+SIN(JP-RAD(.5))*RJ
850 LINE(160,88)-(JX,JY),PSET,3
860 RETURN
```

まず，SCREEN文のパラメータを変化させなければいけませんから，ここに変数を置きます。また，初期設定は表示ページ（出力ページ）と書き込みページ（入力ページ）とを切りかえなくてはなりませんから，表示ページをSC，書き込みページをCSという変数に代入しておきます(160，170行)。最初は１ページ目（出力ページ）が表示されている状態で，２ページ目（入力ページ）に針を書き込みます。書き込みは，180行と190行のGOSUB文で，前の針を消すサブルーチンと新しい針を描くサブルーチンを呼び出すことによって行ないます。このときは１ページ目(出力ページ)，つまり実際に目で見えている画面にはまだ何も描かれていません。

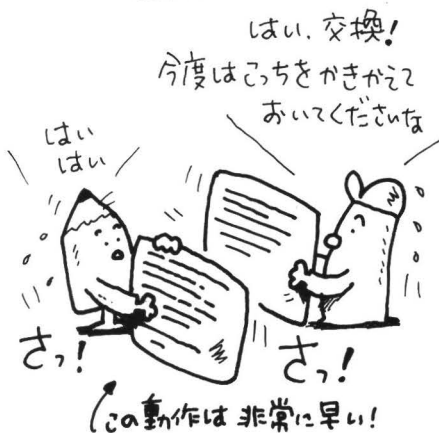
220行でSWAP文によってSCとCSの値を入れかえています。SWAP文は，２つの変数の中身を交換する働きをします。

```
SWAP　変数1，変数2
```

変数１の値を変数２に，変数２の値を変数１に入れて値の交換を行ないます。変数は数値型，文字型どちらでもよいのですが，変数の型は同じでなければならないことはいうまでもありません。

変数SCとCSの中身を入れかえると，表示ページと書き込みページが入れかわります。今まで針の書き込みが行なわれたページが表示され，針が見えるようになります。こうして見えなくなったもう一方のページでは，次の針が描かれています。もう一度220行に来たところで画面が切りかえられ，針が動いたように見えるわけです。前の針を消す部分は，時，分，秒の３本の針を一気に消せるよう，１つにまとめてしまいました。

また秒針の表示を600行から，分針が700行から，時針が800行からにまとめてあり，LINE文による指定によって秒針は黄色，分針は水色，時針はマゼンタで描いています。秒のサブルーチンに少し追加した部分があります。これは，ある１秒の間は，次の書き込みを行なわないようにするためです。この作業を行なうことによって，一時的に針が２本表示されるのを防ぐことができます。時針を描くサブルーチンの中の820行は，秒針，分針よりも少々複雑な計算式を使っています。これは，たとえば５時30分のときに針を５と６の間に持っていくためのもので，どのような処理をしているかをみなさんもプログラムを見ながら考えてみてください。

これで，一応時計の３本の針を動かすことができました。次からは，いろいろなグラフィック命令を使って，時計らしい形を作っていきます。

4-8 パソコンペインタいろいろ

●10進，2進，16進のかずかず

ここでは「色を混ぜる」機能もマスターしていただきましょう。

その前に，グラフィックパターンを作るときにはどうしても必要な知識である2進数と16進数と10進数について説明しておきます。

まず，わたしたちが日常使っている10進数から見ていきましょう。「数字の各位置が10を基数とするベキ乗で示されるような数字の表現方式」というのが10進数の定義ですが，これではよくわかりませんね。この定義を言いかえれば，「0から10になったら桁上がりする」ということです。たとえば，4109という数字は，

$$4109 = 4\times10^3 + 1\times10^2 + 9\times10^0$$
$$= 4\times1000 + 1\times100 + 9$$

というように，(0から9の数)×10のベキ乗で表わすことができます。この10を基数というのです。この10進数はわたしたちの生活にはなくてはならぬものです。ところがコンピュータにとってなくてはならぬものといえば，それは2進数なのです。

2進数とは，先ほどの10進数の定義に従えば「2のベキ乗で表わされる数」ということになります。つまり，0，1と数えていくと次はジュウではなく，イチゼロ（10）になるというわけです。ですから2のベキ乗に書きかえて計算すれば，10進数に変換できます。

$$11110 = 1\times2^4 + 1\times2^3 + 1\times2^2 + 1\times2 + 0\times2^0$$
$$= 16 + 8 + 4 + 2 + 0$$
$$= 30$$

コンピュータのなかでは，10進数で打ち込まれた数字を2進数に直して計算し，またそれを10進数に直して表示しているのです。

16進数もこれと同じように「16のベキ乗で表わされる数」といえます。しかし，数字を表わす文字は0～9の10個しかありませんから，これでは16進数を書き表わすことはできません。これはABCDEFという文字を使うことで解決しています。

10進数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
16進数	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

16進数を10進数に直す方法はもうわかりますね。

$$2B = 2\times16^1 + B\times16^0$$
$$= 32 + 11\times1 = 43$$

$$EF = 14\times16^1 + 15\times16^0$$
$$= 224 + 15\times1 = 239$$

16進数は&Hをつけて10進数と区別しています。HEX$という関数を使えば，10進数を16進数に直してやることができます。ただし注意してほしいのは，HEX$で変換した結果である16進数

は，文字列だという点です。

ちょっと横道にそれましたが，もう一度色の勉強に戻りましょう。

●タイルをならべて色をつける

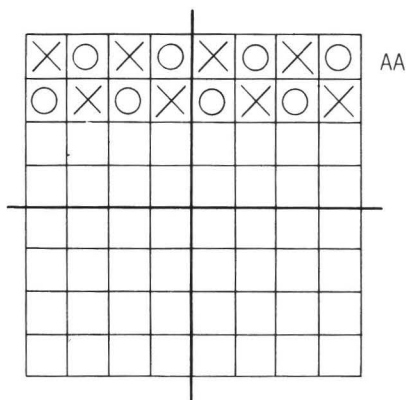
PAINT文で色を塗った部分をよく見てみると，文字や記号やLINE文で描かれた線と同じように，ドットが集まって塗られていることがわかります。X１では，このドットごとに色を指定することによって，基本の８色以外の中間色も作ることができます。

本来，グラフィックスで使われる８色の色は，青，赤，緑という光の３原色の組み合わせで作り出されています（表４－３）。

表４－３　３原色の混合

組み合わせ	結　果
何もなし	黒
青	青
赤	赤
青＋赤	マゼンタ
緑	緑
青＋緑	シアン
赤＋緑	黄色
青＋赤＋緑	白

８ドット×８ドットの枠をまず考えてみましょう。これを**タイルパターン**と呼びます。ちょうどタイルを並べたような形（パターン）です。そしてこのタイル（ドット）のそれぞれに色をつけていきます。たとえば，図４-26のように，Ｂに青を，Ｒに赤を置いたとします。画面は非常に細かいドットの集まりですから，こうすると紫に近い色が得られます。

〔図４-26〕

さて，このパターンの1段目を横に見て，青色のドットがあるところを1，ないところを0とすると，10101010と表わすことができます。1と0の組み合わせで表記された数ですから，これは2進数と考えることができます。これを16進数に直すと，&HAAとなります（「&H」は16進数での表記であることを示す目印です)。同じように1段目の赤色は，&H55になります。緑はまったく使われていないので，2進数で，00000000，16進数では&H00と表わすことができます。この青，赤，緑という光の3原色の順に16進数を並べると，「AA5500」となります。これが，X1のタイリングペイントで使われる色のデータなのです。同じようにして2段目を，〝55AA00〞と表わします。

3段目以降はこの2段の繰り返しですから，あえて指定する必要はありません。したがってドットに赤と青を交互に並べて作り出す紫色のデータは，

```
 1段目        2段目
AA5500      55AA00
青 赤 緑     青 赤 緑
```

となります。

データが用意できましたから，実際にPAINT文で作ってみましょう。

```
10 WIDTH80:SCREEN0,0:CLS4
20 LINE(10,10)-(100,100),PSET,4,8
30 PAINT(11,11),HEXCHR$("AA570055AA00")
40 END
```

このプログラムを実行すると，白で囲まれた枠の中を，紫色で塗りつぶすことができます。

●16進数を切る!?

HEXCHR$関数は，16進数を数としてではなく文字の列として扱いながら，最初から2文字（2バイト）ずつ区切って，その2文字に対応するキャラクタに変換して文字列を作ります。

少しむずかしい話になりますが，巻末のキャラクタコード表を見てください。

この表を使うと，X1が持っているすべての文字や記号を2桁の数字（16進数の表記）で表わすことができます。表の中の「W」の文字を見てみましょう。「W」が書かれた列を上にたどると5，左に行くと7になります。この「57」という数字が，実はアルファベットの「W」を表わします。「♠」の記号を同じように見てみると「E2」と表わされることがわかります。「〒」であれば「FF」です。

「57」も「E2」も「FF」も，すべて2桁の16進数であることはもうおわかりですね。

```
10 A$="CEDEB8C9A05831"
20 PRINT HEXCHR$(A$)
```

このプログラムを走らせると，画面には「ボクノ　X1」と表示されます。HEXCHR$関数が，①A$という16進数の文字列を2つずつに分解し，②その16進数で表わされるキャラクタコードに対応する文字を読み出し，③PRINT文で表示させています。

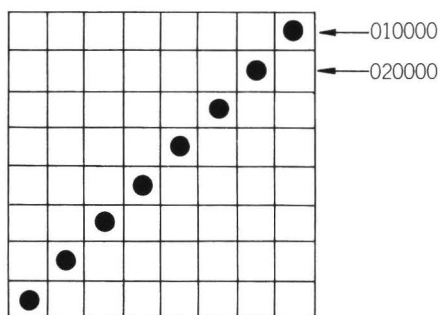
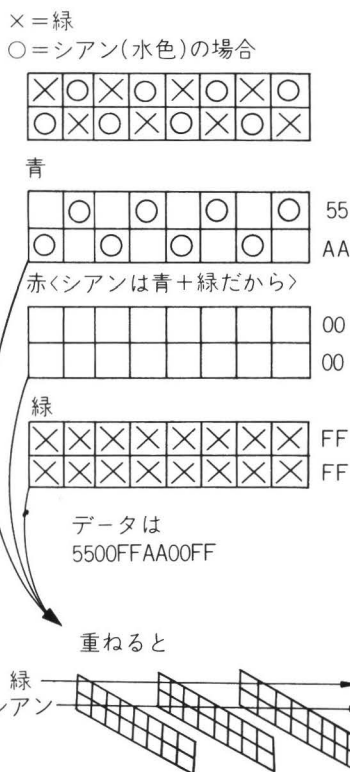
```
CE  DE  B8  C9     A0     58  31
↓   ↓   ↓   ↓      ↓      ↓   ↓
ホ  ゛  ク  ノ  (スペース)  X   1
```

本題に戻りましょう。ここでは，PAINT文に対する色のデータとして16進数を使っていますから，HEXCHR$を使っていても文字や記号が表示されるのではありません。ドットごとに色の指定をして，それを青，赤，緑の３色のそれぞれに８桁の２進数のパターンに直し，さらに２桁の16進数に直すのですから，色のデータを作るのはなかなか面倒な作業です。しかし，ここで学んでいる考え方は，Ｘ１の機能を活かすうえで，大切なものです。じっくりと取り組んでみてください。

●中間色もいろいろ

青，赤，緑の３原色からデータを作ることはわかりましたが，水色（シアン）と緑を交互に置いて若草色を作りたいときはどうすればよいでしょうか。シアンは青＋緑で作られている色ですから，シアンで色を指定するドットには青と緑の両方のデータが置かれなければなりません。図４-27を見てください。

このように，色の３原色に分け，それぞれの色についてデータを調べることになります。

タイリングペイントでは，このような中間色だけでなく，縞模様や綾模様なども作ることができます。たとえば，ななめの縞の場合，１段目の１つの色を青にすれば，16進数の表記で01となります。そうすると青以外の赤，緑のデータは00ですから，１段目は010000，２段目は020000……となります（図４-28）。このようにして，８段分のデータを作って，PAINT文を使えば，いろいろな模様パターンで塗ることもできます。

〔図４-27〕

×＝緑
○＝シアン(水色)の場合
青 55 AA
赤〈シアンは青＋緑だから〉 00 00
緑 FF FF
データは
5500FFAA00FF
重ねると
緑
シアン

〔図４-28〕

010000
020000

次の〔プログラム４-16〕は，赤からはじまって，赤に戻ってくる12色並びのそれぞれの色を塗るものです。各データが，どのようなドットのパターンを作っているか，６文字ずつ区切って調べてみてください。

〔プログラム４-16〕

```
10 WIDTH  80:SCREEN 0,0:CLS 4
20 DIM COL$(12),CNAM$(12)
30 FOR I=1 TO 12
40   READ COL$(I),CNAM$(I)
50 NEXT
100 FOR I=1 TO 12
110   IF (I-1) MOD 8=0 THEN XX=0:YY=YY+1 ELSE XX=(I-1) MOD 8
120   LINE(XX*80,(YY-1)*48+8)-(XX*80+70,(YY-1)*48+40),PSET,7,B
130   LOCATE XX*10,(YY-1)*6:COLOR 7:PRINT CNAM$(I)
140   PAINT(XX*80+1,(YY-1)*48+9),HEXCHR$(COL$(I))
150 NEXT
160 LOCATE 0,12
200 DATA 00FF00,ｱｶ
210 DATA 00FF8800FF22,ﾀﾞｲﾀﾞｲ
220 DATA 00FF5500FFAA,ｷﾀﾞｲﾀﾞｲ
230 DATA 00FFFF,ｷｲﾛ
240 DATA 0055FF00AAFF,ｷﾐﾄﾞﾘ
250 DATA 0000FF,ﾐﾄﾞﾘ
260 DATA 5500FFAA00FF,ｱｵﾐﾄﾞﾘ
270 DATA 5500AAAA0055,ﾐﾄﾞﾘｱｵ
280 DATA FF0000,ｱｵ
290 DATA FF8800FF2200,ｱｵﾑﾗｻｷ
300 DATA FF5500FFAA00,ﾑﾗｻｷ
310 DATA FFFF00,ｱｶﾑﾗｻｷ
```

●データを読む

さてここで，また新しい命令を覚えなければなりません。READとDATAです。READ文は，いつでもDATA文と組になって使われます。そこでDATAという命令を探していると，200行から310行までぎっしりと並んでいます。

DATA文は，文字どおりコンピュータに特定のデータを与える命令です。INPUT文を使うと画面にコンピュータは「？」を表示してデータの入力を要求してきます。そこで必要なデータをプログラム実行中に入力するわけです。DATA文を使うと，あらかじめプログラムの中に必要なデータを書き込んでおくことができるのです。

READ文は，これも文字どおりに「データを読め」という命令です。プログラム中にREAD文があると，プログラムの流れはDATA文を探しにいきます。DATA文が見つかると，READ文の後に書かれた変数に，DATA文で指示されたデータを読み込みます。もちろん数値変数には数値のデータ，文字列変数には文字列のデータというように対応します。プログラムを見ると，DATA文で指示した場合，文字列のデータでも「"」で囲む必要がないということがわかります。

ただし，「,」（カンマ）もデータの一部として使いたい場合は特に「"」で囲みます。このような場合は全部のデータを「"」で囲んでしまう方が見やすいでしょう。

〔プログラム 4 -15〕中では，READ文の後に 2 つの変数を置き，DATA文の後にも 2 つのデータを置き，それぞれの変数とデータを対応させています。変数に対応するデータがないとエラーになります。データが多過ぎることには，問題はありません。

このプログラムではREAD文の後の変数を配列変数とし，FOR〜NEXT文を使って 1 つのREAD文で何回もデータを読み込むようにしています。しばしば使うテクニックですので覚えておくとよいでしょう。

〔プログラム 4 -15〕中では使っていませんが，RESTORE文というものもあわせて覚えておいてください。

READ〜DATAでは，プログラム中ではじめに見つけたDATA文からデータの読み込みをはじめます。そこでRESTORE文を使うと指定した行番号のDATA文からデータの読み込みをさせることができます。また，RESTORE文では行番号だけではなく，ラベルも使うことができます。詳しいことは，後で実際にRESTORE文を使ったプログラムを見ながら確かめることにしましょう。

4-9 時計よ，動け！

●時計の外形を描こう

〔図 4 -29〕

図 4 -29のような時計にしようと思います。

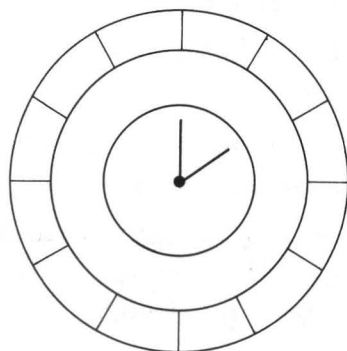
前のところで，12色の色データを作りましたから，時計の外側を時間単位に分割し，それぞれ塗り分けます。また，この色の輪と内側の円の間には，後で文字を入れられるよう，スペースを空けておきます。

例によって，プログラムをいくつかのサブルーチンとしてまとめておくことにします。まず，時計の外形を描くサブルーチンを作っておきましょう(プログラム 4 -17)。OUT-LINEというラベルをつけて使うことにします。CIRCLE文を使って 3 つの円を描きます(880～900行)。パラメータを省略してよいところはすべて省略してしまいました。どのようなパラメータを省略されていて，どのようなパラメータが書き込まれているのか，十分に確認しておいてください。

〔プログラム4-17〕

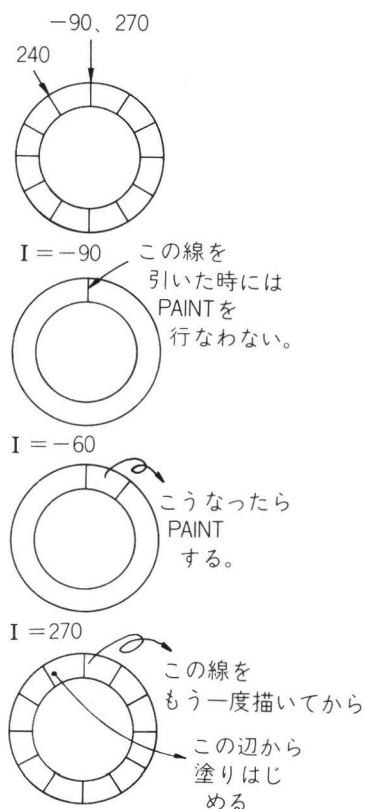
```
870 LABEL "OUTLINE"
880 CIRCLE(160,88),52,4
890 CIRCLE(160,88),76,7
900 CIRCLE(160,88),88
910 RESTORE 1000
920 FOR I=-90 TO 270 STEP 30
930   XG=160+COS(RAD(I))*88:YG=88+SIN(RAD(I))*88
940   XU=160+COS(RAD(I))*76:YU=88+SIN(RAD(I))*76
950   XP=160+COS(RAD(I-5))*80:YP=88+SIN(RAD(I-5))*80
960   LINE(XG,YG)-(XU,YU),PSET,7:IF I=-90 GOTO 980
970   READ P$,A$:PAINT(XP,YP),HEXCHR$(P$)
980 NEXT
990 RETURN
1000  DATA 00FF00,ｱｶ
1010  DATA 00FF8800FF22,ﾀﾞｲﾀﾞｲ
1020  DATA 00FF5500FFAA,ｷﾀﾞｲﾀﾞｲ
1030  DATA 00FFFF00FFFF,ｷｲﾛ
1040  DATA 0055FF00AAFF,ｷﾐﾄﾞﾘ
1050  DATA 0000FF,ﾐﾄﾞﾘ
1060  DATA 5500FFAA00FF,ｱｵﾐﾄﾞﾘ
1070  DATA 5500AAAA0055,ﾐﾄﾞﾘｱｵ
1080  DATA FF0000FF0000,ｱｵ
1090  DATA FF8800FF2200,ｱｵﾑﾗｻｷ
1100  DATA FF5500FFAA00,ﾑﾗｻｷ
1110  DATA FFFF00,ｱｶﾑﾗｻｷ
```

次に，外側の円と内側の円の円周上の点を30度ごとに求め，それをLINE文で結びます。ここでは，前に学んだ円の公式を使って円周上の点の座標を求めます。

図 4 -30を見ると，Iの値が－90のときに12時の位置に縦の線を，－60のときに 1 時の位置に線を引けばよいことがわかります。

970行では，READ文を使って色のデータを読み込んでいます。910行RESTORE文で，「RESTORE1000」というようにデータの読み込み先を指定しています。

データを読み込むと，PAINT文でその色を塗ります。色を塗りはじめる座標点を，950行で計算していますが，座標点はそれぞれの扇形のほぼ中心になるような計算となっています。

このプログラムで注意しておきたい考え方の１つが，950行の後半のIF文による条件判断とその処理です。Ｉの値が－90のときには，READ文を飛び越えてしまうようにしています。

変数Ｉの値が－90の場合，輪を区切る12時の位置の線，つまり１本目の線を描いています。他の区切りがありませんから，PAINT文が実行されると輪の内側の全体が塗りつぶされてしまうことになります。また　図４-30を見るとわかるように，12本の線を引くために，　必要な変数Ｉの値は，－90から240で十分なのですが，　ここではPAINT文を12回うまく働かせるために，　ひとまわりしてきた12時の位置，つまり－270度の位置にも線を描いています。

●時計よ，動け！

いかがでしょうか。プログラムの仕組みがわかったところで，さっそく実行してみましょう。プログラムの途中から実行するときには，GOTO文かGOSUB文を使います。ダイレクトモードで次のように打ち込みます。

```
WIDTH 40 : SCREEN 0, 0 : GOSUB 870
```

うまく動いてきれいな外形が描けたでしょうか。このように，ひとまとめの処理サブルーチンごとにテストすることができます。テストをしてうまく動くようなら，これまでに作ってきた時計プログラムに，このサブルーチンを呼び出す命令をつけ加えて，プログラムを走らせてみることにします。

110行をGOSUB　"OUT LINE"としてやれば，まずこのサブルーチンが呼び出され，後は今までと同じ針を動かす部分を繰り返すわけですから，うまく動きそうです。試してみると，針が動くたびに外形が表われたり消えたりします。この時計では針をスムーズに動かすために，ページを切りかえるマルチスクリーンの方法を使っていたことを思い出してください。１ページ，２ページを交互に切りかえていたので，この外形を両方のページに描いておかなくてはいけなかったのです。SCREEN文で書き込みページを変えて，このサブルーチンを呼び出せばよいわけで，まず，１ページ目に描くためには，

SCREEN, 0：GOSUB "OUT LINE"

次に２ページ目に同じものを描くために，

SCREEN, 1：GOSUB "OUT LINE"

とします。

ここでは外形を描くサブルーチンが２回呼ばれていますのでRESTORE文でデータの読みはじめの場所を指示しておかないと，２度目のときに読むべきデータがなくなってしまうのでエラーとなります。

プログラムを実行すると，最初にこのサブルーチンが呼び出され，時計の形が描かれます。その後はこれまでに作ってきた針を動かすサブルーチンを繰り返します。

ここまでの全体のリストを載せてありますので，全体の流れをよく見ておいてください（プログラム４-18)。

それではプログラムをRUNさせてみましょう。時計よ，動け！

〔プログラム４-18〕

```
100 WIDTH40:CLS4:RB=48:RF=44:RJ=40
110 SCREEN,0:GOSUB "OUTLINE":SCREEN,1:GOSUB "OUTLINE"
160 SC=0:CS=1
170 LABEL "MAIN":SCREENSC,CS
180 GOSUB "HARIREST"
190 GOSUB "HARI秒"
200 GOSUB "HARI分"
210 GOSUB "HARI時":IF FJ=0 THEN FJ=1
220 SWAP SC,CS
230 GOTO 170
500 LABEL "HARIREST":IF FJ=0 THEN RETURN
510 LINE(160,88)-(BXR,BYR),PRESET,6
520 LINE(160,88)-(FXR,FYR),PRESET,5
530 LINE(160,88)-(JXR,JYR),PRESET,3
540 RETURN
600 LABEL "HARI秒"
610 BO=VAL(RIGHT$(TIME$,2)):IF BO=BC GOTO 610
620 BP=RAD(BO*6-90)
630 BX=160+COS(BP)*RB:BY=88+SIN(BP)*RB:BXR=160+COS(BP-RAD(6))*RB
:BYR=88+SIN(BP-RAD(6))*RB
640 LINE(160,88)-(BX,BY),PSET,6
650 BC=BO
660 RETURN
700 LABEL "HARI分"
710 FU=VAL(MID$(TIME$,4,2))
720 FP=RAD(FU*6-90)
730 FX=160+COS(FP)*RF:FY=88+SIN(FP)*RF
740 FXR=160+COS(FP-RAD(6))*RF:FYR=88+SIN(FP-RAD(6))*RF
750 LINE(160,88)-(FX,FY),PSET,5
770 RETURN
800 LABEL "HARI時"
810 JI=VAL(LEFT$(TIME$,2))
820 JP=RAD((JI*30+FU*.5)-90)
830 JX=160+COS(JP-RAD(.5))*RJ:JY=88+SIN(JP-RAD(.5))*RJ
840 JXR=160+COS(JP-RAD(.5))*RJ:JYR=88+SIN(JP-RAD(.5))*RJ
850 LINE(160,88)-(JX,JY),PSET,3
860 RETURN
870 LABEL "OUTLINE"
880 CIRCLE(160,88),52,4
890 CIRCLE(160,88),76,7
900 CIRCLE(160,88),88
910 RESTORE "ｲﾛﾃﾞｰﾀ"
920 FOR I=-90 TO 270 STEP 30
```

```
930    XG=160+COS(RAD(I))*88:YG=88+SIN(RAD(I))*88
940    XU=160+COS(RAD(I))*76:YU=88+SIN(RAD(I))*76
950    XP=160+COS(RAD(I-5))*80:YP=88+SIN(RAD(I-5))*80
960    LINE(XG,YG)-(XU,YU),PSET,7:IF I=-90 GOTO 980
970    READ P$,A$:PAINT(XP,YP),HEXCHR$(P$)
980 NEXT
990 RETURN
1000 LABEL "ｲﾛﾃﾞｰﾀ"
1010  DATA 00FF00,ｱｶ
1020  DATA 00FF8800FF22,ﾀﾞｲﾀﾞｲ
1030  DATA 00FF5500FFAA,ｷﾀﾞｲﾀﾞｲ
1040  DATA 00FFFF00FFFF,ｷｲﾛ
1050  DATA 0055FF00AAFF,ｷﾐﾄﾞﾘ
1060  DATA 0000FF,ﾐﾄﾞﾘ
1070  DATA 5500FFAA00FF,ｱｵﾐﾄﾞﾘ
1080  DATA 5500AAAA0055,ﾐﾄﾞﾘｱｵ
1090  DATA FF0000FF0000,ｱｵ
1100  DATA FF8800FF2200,ｱｵﾑﾗｻｷ
1110  DATA FF5500FFAA00,ﾑﾗｻｷ
1120  DATA FFFF00,ｱｶﾑﾗｻｷ
```

4-10 プログラムの構造とプログラミングのコツのお話

●プログラムの構造とフローチャート

これまでに，いくつかのプログラムを組み立ててきました。プログラムを１つ１つの部分について，細かく分析しながら作ってきたわけですが，各部分の働きはよく理解できたことと思います。ところが，組み上がったプログラムを全体として眺めると，どこがどのような働きをしているのか，一見しただけでわかりにくくなっているかもしれません。特に，GOTO文でプログラムの流れを変えているところなどは，どのようなときに分岐するのか，理解しにくいものです。

このようなときには,プログラムの流れを図式化して表わしておくと，後から手を入れる場合などに便利です。

図式化のためによく使われるのが「フローチャート（流れ図）」と呼ばれるものです。もっとも一般的なフローチャートは，図4-31のような記号を使ってプログラムの流れを書き表わすものです。

図の例のように，これらの記号の中に主な処理の内容を記入し，さらに記号の右上にそれぞれの処理に該当する行番号を書いておくとよりわかりやすくなります。

パソコンのユーザーが自分のプログラムを図式化する場合には，これらの記号を厳密に使い分けなくてはいけない,というのではありません。誰でも何かのプログラムを作ろうと考えはじめたときから，全体の流れが少しずつ頭の中でまとまってきているものです。その全体の流れを，このような記号を基本に，わかりやすくまとめておけばよいのです。

〔図4-32〕

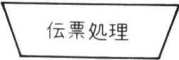
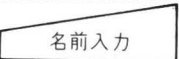
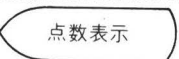
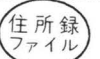
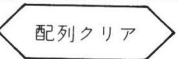
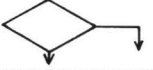
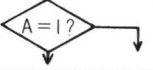
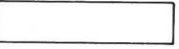
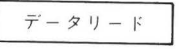
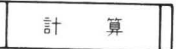
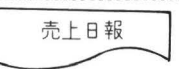
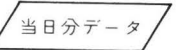
START

配列宣言

"ショキカ"

"ニュウリョク"

時刻表示

"コントロール"

例として，「テレビ24時間予約プログラム」の大まかな流れをフローチャートに書き表わしてみました（図4-32)。これを見るかぎり，大変すっきりしたプログラムに見えます。プログラムの流れというのは，フローチャートに表わしたときに，このように上から下へ，という一連の流れであるのが望ましいのです。ところが，このプログラムで一番大きな部分を占めている「ニュウリョク」という部分を詳細にフローチャート化してみると，図4-33のように，IF〜GOTO文による流れの変化が，複雑にからみあい，見にくいものになっていることがわかります。ここで，プログラムの構造について考えてみましょう。

まずはじめに，なぜこのようなプログラムになってしまうのかを考えてみましょう。たとえば，時，分などの入力時に不都合なデータの入力をさける処理を後からつけ足し，GOTO文によって戻しているためだ，ということをあげることができます。このGOTO文こそが，プログラムの流れを複雑にしている原因なのです。特にこのプログラムのように，大まかな骨組を作ってから必要な補助的処理を加えていくという作業を何回も繰り返していると，最初のプログラムがどんなにすっきりしたものであったとしても，最後にできあがったときは複雑なものになっ

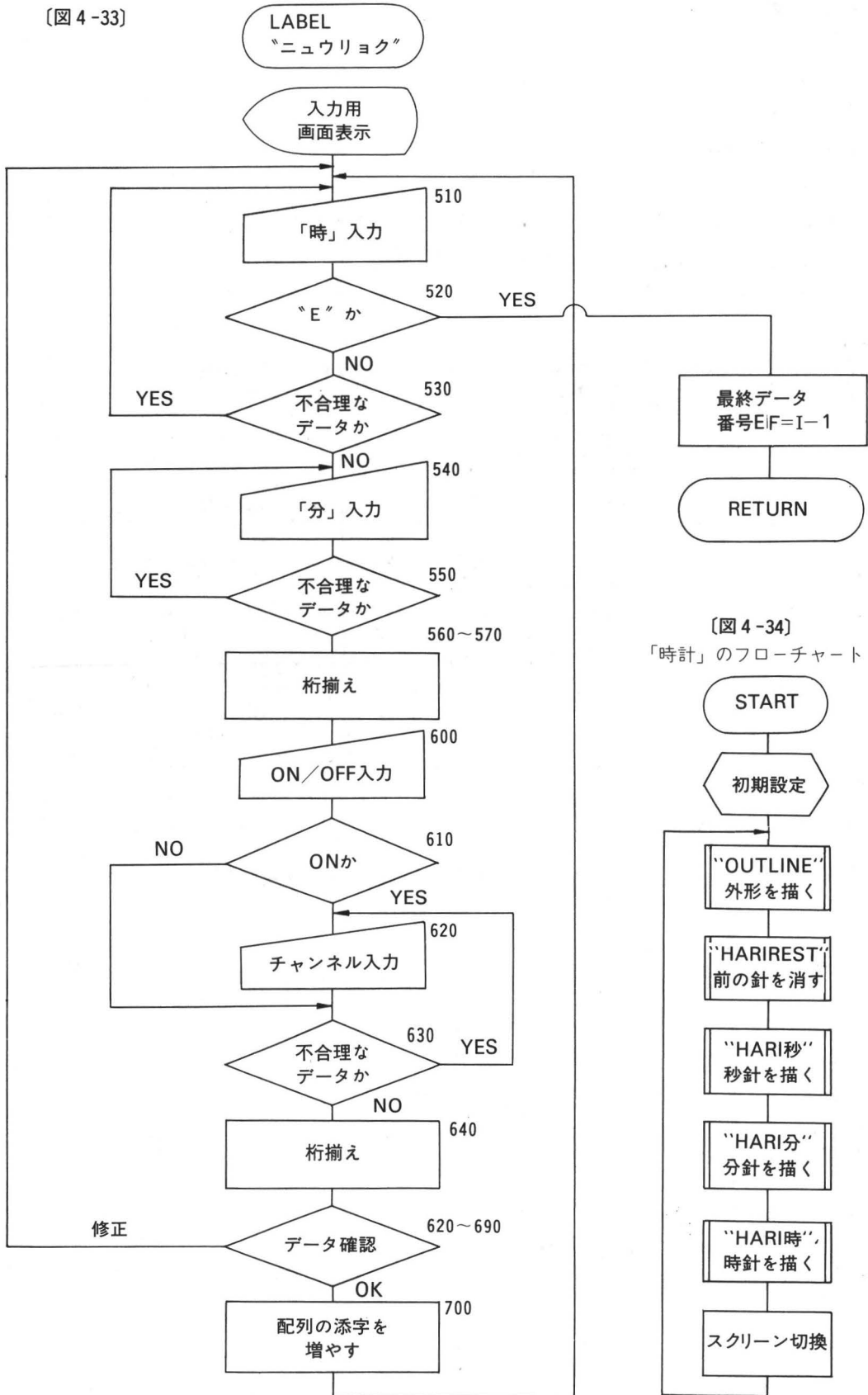
〔図4-31〕フローチャートに使われる記号

記号	名称	内容	例
	端子	開始や終了，停止など，流れの端部を表わす	START
	手作業	コンピュータで処理する以前のデータの整理など，手作業による処理（あまり使わない）	伝票処理
	手操作入力	機器を操作して入力を行なう。普通はキーボードからの入力を示す	名前入力
	表示	CRTディスプレイやランプによる出力表示を表わす	点数表示
	磁気テープ	ファイルの入出力などテープを使った処理を表わす	住所録ファイル
	準備	初期設定や初期化を表わす	配列クリア
	判断	条件判断による分岐などを表わす	A=1?
	処理	あらゆる内部処理を表わす	データリード
	定義済処理	主にサブルーチンを一括して表わす時に使う	計算
	書類	プリンタによって出力される書類を表わす	売上日報
	入出力	あらゆる入出力操作を一般的に表わす	当日分データ
	磁気ディスク	主にフロッピーディスクによる入出力を表わす	顧客データ
	結合子	フローチャートが大きくなり分割される時に分割する点に使い，中につながりがわかるよう記号を入れる	1 1

てしまいます。

今度は，「時計」のプログラムをフローチャートに表わしてみましょう(図4-34)。こちらは，大変すっきりとした，上から下への流れを持った構造になっています。

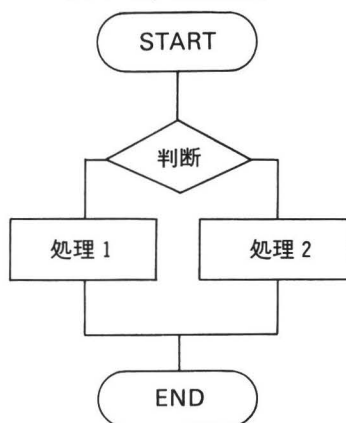
各サブルーチンの内容を見ても，GOTO文は全体でわずか3カ所しかありません。そのうちの1つは，全体としての流れを繰り返すためのものであり，他の2つは不要な処理を飛び越すために使われているだけです。

〔図4-33〕

〔図4-34〕
「時計」のフローチャート

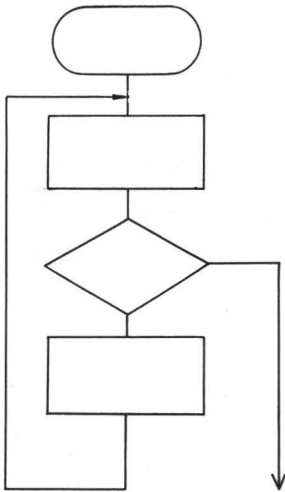
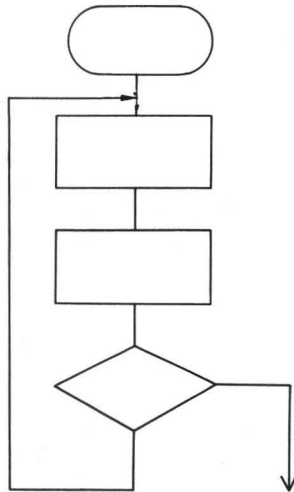
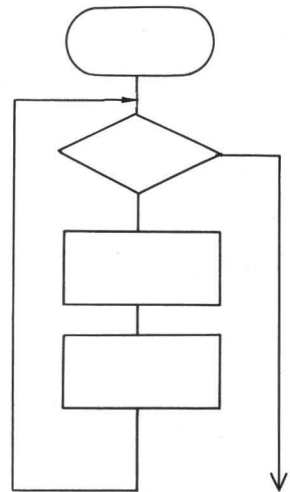
●構造化プログラミングのテクニック

このように，流れや処理内容の理解しやすいプログラム作りのために，「構造化プログラミング」という技法があります。構造化プログラミングは，GOTO文を原則として使わないというのが基本的な考え方になっています。それでは，どのような考え方に基づいてプログラムを組み立てるのでしょうか。

構造化プログラミングでは，プログラムを組み立てる基本構造として，3つに分類しています。まずその1つは，図4-35のように，処理の流れが次々と下に向けて進んでいく「連接」の構造です。プログラムの流れは，このようになるのが最も望ましい構造です。

〔図4-35〕連接構造　〔図4-36〕選択構造

2つ目は，何かの条件によって，処理がいくつかに分かれる「選択」構造です（図4-36）。この場合，どちらの処理を選択しても，再び1つの流れに戻り，全体としてはやはり上から下への流れを保っています。これは，IF（判断）THEN〔処理1〕ELSE〔処理2〕という流れに対応しています。ON～GOTOやON～GOSUBによる2つ以上の分岐も，この考え方としてとらえることができます。

3つ目は，「繰り返し」構造です。「時計」のプログラムでは，いつまでも繰り返しを続ける「無限ループ」になっていますが，多くの場合，何かの条件によって繰り返しから抜け出すことが必要です。これらの繰り返しからの抜け出し方については，図4-37のような3とおりの方法が考えられます。(1)のような方法は，ループの構造としては不自然です。特に，内部での変数の処理が，理解しにくくなります。(2)と(3)は，ループの先頭と終わりで判断する方法です。(2)の形式は，X1のBASICで言えば，REPEAT～UNTILが対応します。この構造は，どのような場合でも1回はループの中を通らなければなりません。(3)の形は，もっとも合理的と考えられる方法です。なぜなら，必要がなければ一度もループを通過しないですむからです。これは，WHILE～WENDに対応します。

〔図4-37〕

●プログラムのコマギレ

以上3つの基本構造を守って作られたプログラムは，大変すっきりとした構造を持ち，理解しやすいものになるはずです。このようなプログラムの構造化に有効なのが，プログラムのモジュール化です。モジュール化というのは，いくつかの部分品に分けたまとまりから全体を作るという意味です。

「時計」のプログラムでは，それぞれのサブルーチンが独立した働きを持っており，これらを部分品と呼ぶことができます。サブルーチンを順に呼び出していくことによって，上から下へと流れる，連接構造（全体として繰り返す）になっています。プログラムをモジュール化することの利点はいくつかあります，その1つは，モジュールごとの機能をテストすることができるという点です。これまでにも，GOTO文を使ってモジュールのテストをしてきました。これらのプログラムがさらに大きくなって，10倍もの長さのプログラムを作る場合を考えてみると，モジュール化の利点がよくわかります。たいていの人は，そのような長いプログラムを作ろうと考えたとしても，一体どこから手をつけていいのかわからなくなってしまうと思います。このようなときに，1つ1つの処理をモジュール化して作っていくと，各モジュール自体は短いものになりますから，少しずつまとめながら作っていくことができます。このようにして組み上げられたプログラムは，各モジュールごとの動作チェックが完全に行なわれていれば，全体として大きな誤りは発生しません。だらだらと長く作られたプログラムでは，誤りを発見しようとしてもむずかしくなります。全体を通して誤りをさがさなければならないからです。モジュール化されたプログラムでは，その処理を行なっているモジュールごとに再チェックすることで原因を追究できます。

一般には，プログラムを作りはじめるときには，まずそのプログラムの中心となる部分や，もっともむずかしそうな部分から作っていき，最後に細かい部分を手直ししていく，という順になることが多いようです。実際には，この細かい部分の手直しなどが，デバッグと並んで最

も手間と時間のかかる作業になり，結果的に能率の悪いプログラム作りになってしまうわけです。構造化プログラミングではモジュール化を行なうことで，全体の流れに即した合理的なプログラム作りを行なうことができます。

この本では，BASICの解説書という性格上，掲載されているプログラムすべてがこの構造化プログラミングの手法によって作られているわけではありませんが，少なくともモジュール化の方法は極力取り入れて作ってあります。これから自分でプログラム作りに挑戦しようとする気持ちを持っている人は，**「GOTO文を原則として使わない」「プログラムをモジュール化する」「上から下への流れを尊重し，この流れに即したプログラム作り」**の3つを基本とした，構造化プログラミングの方法をおすすめします。

4-11 文字，いろいろ

●アスキーコードの秘密

プログラミングから少し離れて，X１の持っているいろいろな文字や記号を見ておきましょう。

これまでは，アルファベットとカタカナ，それに数字を主な記号として使ってきましたが，X１は他にもいろいろな文字や記号を持っています。前に少しだけ説明しましたが，漢字もいくつかありますし，「〒」マークも用意されています。まずX１がどんな文字を持っているか，全部の文字を面画に表示させてみましょう。

```
FOR 1=32 TO 255 : PRINT CHR$(I) ;: NEXT
```

巻末のアスキーコード表を見てください。ここには，X１が持っているすべての記号や文字があります。文字のそれぞれに，いわば背番号のようなものがついていることは，前にお話しました。文字や記号は，アスキーコードの32から255に入っていることが表からわかります。つまり文字や記号は，32から255の背番号を持っているということです。これ以下のコードには，コントロールコードが入っています。たくさんあるコントロールコードを表４-４にまとめてみました。

試しに[CTRL]キーを押しながら[G]キーを押してみましょう。表にもあるようにベルが鳴ります。[CTRL]＋[L]を押すと，画面の表示を消去します。つまり，CLS文と同じ働きをします。プログラム中でPRINT CHR$(12)またはPRINT CHR$(&H0C)とすると，[CTRL]＋[L]の働きをします。表から，Lの出力コードは，16進数で０C，10進数で12であることがわかります。

次の例は，指定された以外のデータが入力されたときに，再入力をさせるプログラムです。[CTRL]＋[E]に相当する命令を，CHR$(5)（カーソルから後を１行消す）を使って，前に入力したデータが画面上に残らないようにしています。コントロールコードをプログラム中で使うと，とても便利なことがわかりますね。

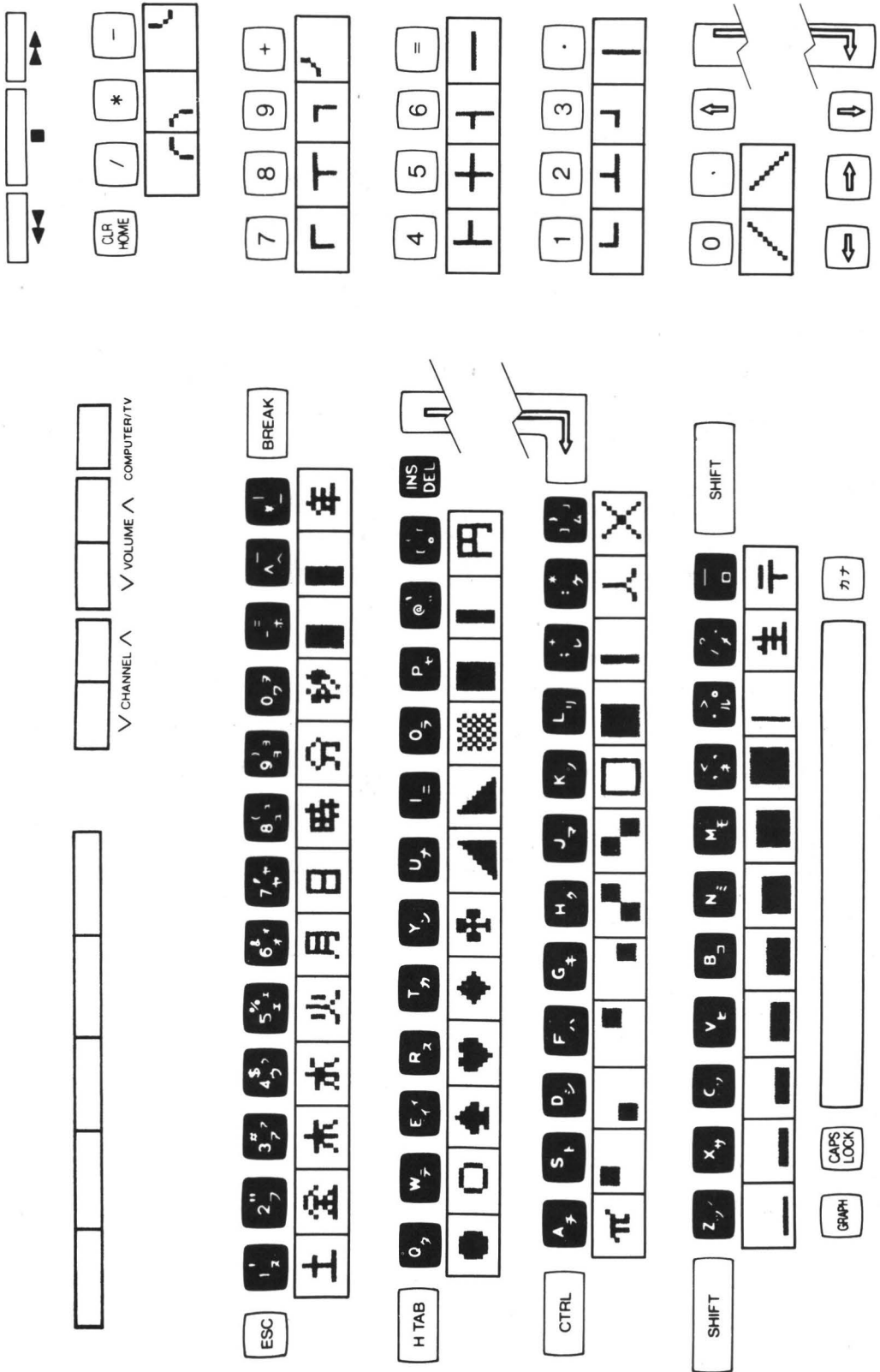
●グラフィック記号いろいろ

X１の持つ文字のうち，キー上に表示されていないものは，[GRAPH]キーを併せて押すことによって表示されます。どのキーにどの文字が割り当てられているかは，図４-38のグラフィックキー配置図を見ない限りわかりません。

[CTRL]と[GRAPH]キーをいっしょに押しながらメインキーを押すと，カーソルの位置は動かず，その位置で押されたキーに対応したグラフィック文字が次々表示されます。希望の文字記号を見つけ出したところで，[CTRL]キーを戻し，[GRAPH]キーとそのキーを併せて押すと希望のグラフィック文字を入力することができます。表の枠を書くときによく使う記号は，テンキーの部分に集められていますから，これも覚えておくとよいでしょう。

表4-4　コントロールコード表

CTRL +	出力コード 16進	出力コード 10進	処　理　内　容	参　考
@	0 0	00		
A または a	0 1	01	INSTモードの設定・解除をする	
B　b	0 2	02	カーソルを1ワード分左へ戻す	
C　c	0 3	03	実行を停止する	SHIFT+BREAK
D　d	0 4	04	スクリーンモードなどを初期状態にする	INIT
E　e	0 5	05	カーソルから右を行の終わりまで消す	
F　f	0 6	06	カーソルを1ワード分右へ進める	
G　g	0 7	07	ベルを鳴らす	
H　h	0 8	08	文字を抹消する	DEL
I　i	0 9	09	水平TAB(スペース出力) の実行	HTAB
J　j	0 A	10	ラインフィードをする	
K　k	0 B	11	ホームポジションへカーソルを移動する	HOME
L　l	0 C	12	画面を消去する	CLR
M　m	0 D	13	キャリッジリターンをする	
N　n	0 E	14	カーソル行から上を上方向へスクロールする	
O　o	0 F	15	〃　下を下方向へ　〃	
P　p	1 0	16		
Q　q	1 1	17	一時停止を解除する	
R　r	1 2	18	空白を挿入する	INS
S　s	1 3	19	一時停止をする	BREAK
T　t	1 4	20	水平TABを設定する	
U　u	1 5	21		
V　v	1 6	22		
W　w	1 7	23	次の行と結合する	
X　x	1 8	24		
Y　y	1 9	25	水平TABを抹消する	
Z　z	1 A	26	カーソル以下の画面をすべてクリアする	
[	1 B	27		
¥	1 C	28	カーソルを右へ移動	→
]	1 D	29	カーソルを左へ移動	←
^	1 E	30	カーソルを上へ移動	↑
−	1 F	31	カーソルを下へ移動	↓
0			画面の背景色を黒にする	COLOR , 0
1			〃　青　〃	〃　1
2			〃　赤　〃	〃　2
3			〃　マゼンタ　〃	〃　3
4			〃　緑　〃	〃　4
5			〃　シアン　〃	〃　5
6			〃　黄色　〃	〃　6
7			〃　白　〃	〃　7

〔図4-38〕グラフィックキー配置図

●文字が反転

X１では，これらの文字の表現方法をいくつか指定することができます。

まず，CTRL とテンキーの－を一緒に押してみましょう。これだけでは画面には何の変化も見られません。しかし，キーボードから何文字か打ち込んでみると，それらの文字が白い四角の中に黒く抜けたような形で表示されます。これを，文字の**反転モード**といいます。この状態で CLR を押すか CTRL ＋ L を押すと，画面全体が反転してしまいます。

画面に何も表示されていない状態というのは，空白の文字〔CHR$(32)〕で画面全部が埋めつくされた状態ということもできます。反転モードでは，この空白も反転しています。ですから，反転モードで CLR を押すと画面全体が白くなります。反転モードを元に戻すには，もう一度 CTRL ＋－を押します。

これらの操作をプログラム中で行なうには，CREV命令を使います。CREVは，キャラクタリバースの略で，後に１をつけると反転モードに，０をつけるか省略するとノーマルモードに戻ります。COLOR文で文字に色指定をしてある場合，色の反転は表４－５のように，補色関係になります。

表４－５　補色関係

青	←→	黄
赤	←→	シアン
緑	←→	マゼンタ
白	←→	黒

反転文字は，画面表示の一部分にアクセントをつけたいときなどに使用すると効果的です。次の〔プログラム４－19〕を入力してみて反転の様子を見ておきましょう。

〔プログラム４－19〕

```
10 FOR C=0 TO 7
20 COLOR C
30 CREV1:PRINT"A";
40 CREV0:PRINT"A"
50 NEXT
```

●文字が点滅

今度は，CTRL とテンキーの＊を一緒に押してから今と同じような操作を試してみましょう。今度は打ち込んだ文字が**反転モード**と**ノーマルモード**を交互に繰り返しています。これを**点滅モード**といいます。色の関係は，反転モードと同じで，補色関係になっています。

プログラム中で使うステートメントは，CFLASHです。オペランドに書くパラメータはCREVと同様，１で点滅モードに，０または省略でノーマルモードに戻ります。点滅モードの場合も CLR を押すと画面全体が点滅状態になります。CFLASHによる点滅は，カーソルの点滅と反対のタイミングになります。カーソルの■がついているときはCFLASHで書いた文字はノーマルに表示されるということです。INKEY$(1)と組み合わせて，入力すべきデータをわかりやすく表示したのが〔プログラム４－20〕です。

〔プログラム 4 -20〕

```
10 LOCATE 10,12
20 PRINT "SELECT   1  or ";:CFLASH 1:PRINT "2"
30 LOCATE 19,12:REPEAT:A$=INKEY$(1):UNTIL A$="1" OR A$="2"
40 CFLASH
```

点滅モードの文字表示は，反転モードよりもさらに目立ちますから，画面に説明文を表示するときや，特に注意を要する表示部分に使うときわめて効果的です。また，プログラムの流れが，長い計算を行なっている間や，データの整理をしている間など普通は画面表示に変化を与えることができず単調になりがちです。このようなときに点滅文字で，メッセージを出しておけば，画面に変化を持たせることができますから，みなさんも使ってみてください。

〔プログラム 4 -21〕

```
10 CLS
20 LOCATE8,12
30 CFLASH:COLOR6:PRINT"ｹｲｻﾝﾁｭｳ ｼｮｳｼｮｳ ｵﾏﾁｸﾀﾞｻｲ !"
40 CFLASH
50 FOR I=1 TO 1000
60 S=S+I
70 NEXT
80 LOCATE0,11:CFLASH:COLOR5:PRINT"ｵﾏﾀｾｼﾏｼﾀ｡"
90 PRINT"1 ｶﾗ 1000 ﾏﾃﾞﾉ ｺﾞｳｹｲﾊ ";S;"ﾃﾞｽ｡"
```

●文字が大きくなった！

プログラムの説明文などを表示するときに，タイトルに大きな文字を使えると便利です。X 1 では，これまで使ってきた標準サイズの文字の他に，3 種類の大きさで文字を表示することができるのです。

文字のサイズの指定は，CSIZE（キャラクタサイズの略）という命令で決めます。CSIZEのオペランドには，0 から 3 までの数値を書きます。

0 を指定するとノーマルサイズの文字になります。

1 では文字の縦の長さが 2 倍，つまり 2 行にわたって縦長の文字表示されます。

2 では横幅が 2 倍の文字となります。

3 を指定すると縦，横とも 2 倍の大きな文字になります。

これらの拡大文字を表示するには，普通のPRINT文ではなく，PRINT #0を使います。もちろんCOLOR指定やCREV，CFLASHも組み合わせて使うこともできます。ただし，使い方にいくらかの制約があります。画面をレイアウトするときは，次にあげる制約に注意してください。

1．CSIZE 1のとき，文字の表示が 2 行にわたりますが，その 2 行にノーマルサイズの文字やCSIZE 2で指定した横長の文字を書いてはいけません。CSIZE 1で指定した文字が，正常に表示されなくなります。

2．CSIZE 2を指定したときには，ノーマル文字が同じ行にあってもかまいません。カーソル位置のX座標は必ず偶数にしなければなりません。

3．CSIZE 3のとき，2 文字× 2 行文の大きさになります。つまり縦 2 行にわたって表示されるのですが，その 2 行の中にノーマル文字とCSIZE 2で指定する横の文字があってはいけません。また，カーソル位置のX座標は偶数でないと正常に表示されなくなります。

CSIZEで文字の大きさを変えて表示を行なった場合，必ずCSIZE 0，またはCSIZEで元に戻

しておきます。元に戻すのを忘れると後の表示に異常が起きることがありますから注意してください。

-らんだむめも-

プログラムを止めたり動かしたり…

まとまったプログラムを組み上げていざ走らせてみたら，エラーで止まりはしないものの，実行された結果がなんとなくおかしい，ということがよくあります。このようなときに，プログラムの途中で一度止めて，変数の値を調べてみると，比較的簡単に原因をみつけることができます。このようなときに便利なのがSTOP文です。

変数の内容が変化する部分の前後にSTOP文を入れておくと，「Break ｉｎ 行番号」と表示してプログラムの実行は止まります。そこで，PRINT文をダイレクトモードで実行し，変数の値を調べることができます。プログラムの実行を再開するにはCONTを使います。CONT文を実行すると，STOP文のあった次のステートメントからプログラムの実行を再開します。マルチステートメントの間にSTOPをおいた場合，次のステートメントから再開させることができ，また，STOP文でプログラムが止まったときに変数の内容変更もできます。

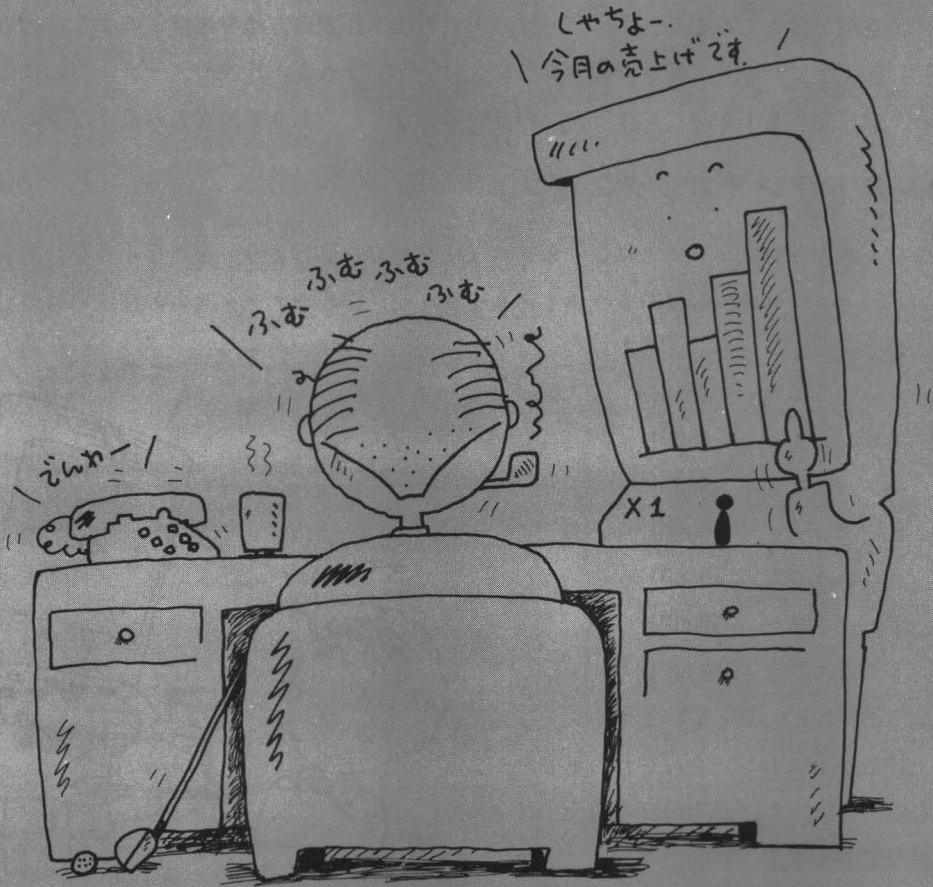
CONT文は，[SHIFT]＋[BREAK]でプログラムを止めたときにも使用することができます。ただし，プログラムの修正などを行なった場合には，CONTで実行を再開できないこともあります。X1では，CONT文でプログラムの再開ができるときには，「Ok.」というようにピリオドがついて表示されます。再開できないときには，「Ok」とだけ表示されます。

このような方法で，変数の内容を確かめ不合理な点を見つけ出した場合には，その変数を変化させている部分を探して調べることになります。このときにはSEARCH文を使うと便利です。たとえば，Aという変数の値がおかしいときに，

SEARCH "A="

とすると，「A=」のついた文のある行がすべて表示され，プログラム全体を通じて能率よく調べることができます。またSEARCH文では，どのような文字列でも探すことができます。たとえば「LABEL」を探すことで，サブルーチンのはじまる行番号を調べることもできます。

5

ビジネスセクレタリーX1

5-1　データの整理と記入〔作表〕

これまでX１のいろいろな機能と，BASICの基礎をみなさんは学んできました。BASICに対する理解も進んだことでしょうし，これからはビジネス，ゲーム，学習などのプログラムを見ながら，具体的なプログラムの組み方を見ていくことにしましょう。

X１をビジネスに使おうと考えている方も少なくないでしょう。

ビジネス用のプログラムで行なう作業の中心は，統計と分析，そしてデータの整理です。そして忘れてはならないのが，入力されたデータをきれいな表にまとめる作業です。いくら正しく入力されたデータでも，それをうまく利用できるように表示（出力）されなければ，データとしての価値は半減してしまいます。

ここでは，小さなレストランをモデルにして，１日の売上げ量から，売上げ金額を計算するプログラム例を考えてみましょう。データの入力方法と作表の方法を理解するように努めてください。X１をビジネスに使うつもりはないよ――という人も，作表やデータ入力の技法はいろいろと役に立ちますから，レストランの経営者になったつもりで読んでみてください。

●レストラン「エックスワン」

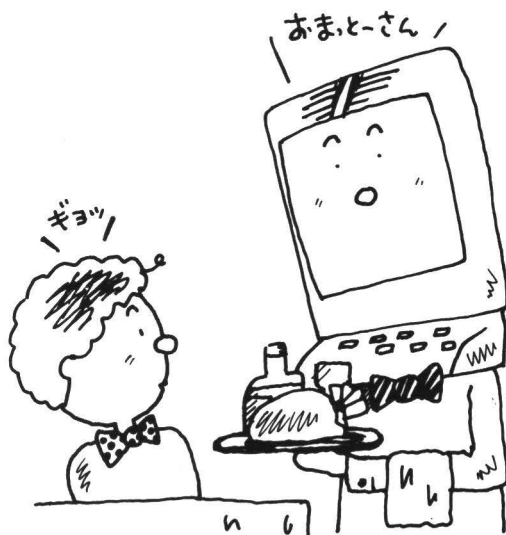
小さなレストラン「エックスワン」のメニューを見てみましょう。小さな小さなレストランですので，メニューの内容はあまり多くありませんが，作表やデータ入力の技法の練習ですから，このぐらいでいいでしょう。

１．カレーライス　　450円
２．エビフライ　　850円
３．ハンバーグ　　800円
４．スパゲッティ　　500円
５．カニグラタン　　960円
６．ポークソテー　1000円
７．ミックスフライ　880円

各商品の単価がわかっていますから，１日の売上げ数量を入力し，それぞれの単価をかけたものを合計すれば，その日の総売上げ金額が計算できます。

品名と単価はDATA文で用意しておきます。

おなじみになったFOR～NEXTとREAD文でこれらのデータを読み込むようにします。10行～80行のサブルーチンが，DATA文からデータを読み込む部分です。

単価を変更したときなど，DATA文の中を修正するだけで簡単に対応できます。

〔プログラム5-1〕

```
10 WIDTH80:SCREEN0,0:CLS4
20 GOSUB"ﾃﾞｰﾀ"
30 GOSUB"ｶｽﾞ"
40 GOSUB"ｶｸﾆﾝ"
50 GOSUB"ﾋｮｳ"
60 END
70 LABEL"ﾃﾞｰﾀ"
80 RESTORE550:FOR I=1 TO 7
90 READ MENU$(I),NEDAN(I):NEXT
100 RETURN
110 LABEL"ｶｽﾞ"
120 PRINT "ﾋﾝﾒｲ            ﾀﾝｶ          ｶｽﾞ"
130 PRINTSTRING$(30,"-"):CONSOLE2,23
140 FOR I=1 TO 7
150 LOCATE0,I+1:PRINT USING"&          &";MENU$(I);
160 PRINT USING"###,### 円";NEDAN(I),
170 LOCATE28,I+1:INPUT"",KAZ(I)
180 NEXT
190 RETURN
200 LABEL"ｶｸﾆﾝ"
210 FOR I=1 TO 7
220 LOCATE0,I+1:PRINT USING"&          &";MENU$(I);
230 PRINT USING"###,### 円";NEDAN(I),
240 LOCATE27,I+1:PRINT KAZ(I)
250 NEXT
260 LOCATE0,12:PRINT "ﾃｲｾｲ ﾊ ｱﾘﾏｽｶ? (Y/N) ";
270 YN$="YyNn"
280 REPEAT:A$=INKEY$(1):AN=INSTR(YN$,A$):UNTIL AN<>0
290 IF AN>2 THEN RETURN
300 LOCATE28,2:INPUT"", DUMMY$
310 FOR Y=2 TO 8
320 KAZ(Y-1)=VAL(SCRN$(27,Y,5))
330 NEXT
340 GOTO 210
350 LABEL"ﾋｮｳ"
360 WIDTH80:CLS
370 LOCATE0,0:CSIZE2:PRINT#0"ｳﾘｱｹﾞﾋｮｳ"
380 PRINT"┌─────────┬───────┬────────┬──────────┐"
390 PRINT"│ ﾋ ﾝ ﾒ ｲ  │ ｶ  ｽﾞ │ ﾀﾝｶ(円) │  ｷ ﾝ ｶﾞ ｸ  "
400 BO$="├─────────┼───────┼────────┼──────────┤"
410 TTL=0
420 FOR I=1 TO 7
430 PRINTBO$
440 PRINT"│";USING"&          &";MENU$(I);
450 PRINTTAB(11);"│";TAB(14);USING"#,###";KAZ(I);
460 PRINTTAB(20);"│";TAB(23);USING"#,###";NEDAN(I);
470 PRINTTAB(29);"│";TAB(32);USING"#,###,###";KAZ(I)*NEDAN(I);
480 PRINTTAB(42);"│"
490 TTL=TTL+KAZ(I)*NEDAN(I)
500 NEXT
510 PRINT BO$
520 PRINT "  ｺﾞ ｳ ｹ ｲ                        ";TAB(31);USING"##,###,
###";TTL;
530 PRINTTAB(42);"│":PRINT"└─────────┴───────┴────────┴───────────
──────┘"
540 RETURN
550 DATA ｶﾚｰﾗｲｽ,450,ｴﾋﾞﾌﾗｲ,850,ﾊﾝﾊﾞｰｸﾞ,800,ｽﾊﾟｹﾞｯﾃｨ,500,ｶﾆｸﾞﾗﾀﾝ,
960,ﾎﾟｰｸｿﾃｰ,1000,ﾐｯｸｽﾌﾗｲ,880
```

●書式指定

それぞれの数量を入力するのが,「カズ」というラベル名でまとめられた110行～190行のサブ

ルーチンです。ここでは、タイトル部分を誤操作によって消してしまうことを防ぐためにCONSOLE文を使っています。入力方法は，FOR〜NEXT文を使って，まずDATA文から読み出した品名を表示し，次にそれぞれの商品の売上げ数量を入力するようにしてあります。入力方法をわかりやすいものにするというのも，プログラムを考えるうえで大切なことです。プログラムを作った人だけが操作するわけではありませんから，データの入力ルーチンの作り方に工夫をしなければなりません。

150行ではまず品名を表示しますが，ここではPRINT USINGという**書式指定**を利用しています。**書式指定**というのは，画面に文字や記号などを表示させるときの形式の指定です。USING "&　　　　&" を使うと，文字列の長さがいくつあっても，&から&までの長さで表示されることになります。この場合，文字列の方が&から&までの長さよりも長ければ残りの文字列は無視され，短ければ足りない分スペースが足されます。150行の場合は，&と&の間，スペースは８文字分ですから，&の２文字を加えた10文字分だけを表示することになります。

230行は，それぞれの商品の単価を表示する部分です。ここでもPRINT USINGが使われています。"＃＃＃，＃＃＃"は，桁数６桁で３桁目にカンマを入れる指定です。金額等を表示する場合，帳簿でもよく使われるように３桁ごとにカンマを入れると読みやすくなります。これらの＃や&を**書式指定子**と呼びます。書式指定の種類を次に挙げておきましょう。

1.「＃」　対象：数値

表示したい最大桁数を指定するものです。表示したい希望の数だけ＃を並べます。桁数が指定より小さい場合には，文字列の前に足りない分だけスペースがつけ加えられ，右づめで表示されます。このようにしておくと，ただ数値を入力しただけでも，位取りの桁が揃うというわけです。桁数が指定よりも多いとエラーが出てストップします。

2.「.」　対象：数値［例］＃＃．＃＃

＃と組み合わせて，小数点位置と小数点以下の桁数を指定します。小数点の位置をそろえて右づめで表示します。このとき，端数は四捨五入されて表示されます。ただし，四捨五入されるのは表示のうえだけですから，数値を計算に使う場合には注意しなければなりません。

4.「＋」,「－」　対象：数値　［例］＋＃＃，＃＃－

正負の符号をつける位置を指定します。＃の後に「＋」「－」の記号をつけると，数値が表示されるときにもその後に符号がつきます。＃の前に置くと，普通と同じく，数値の前に記号がつきます。

5.「＊＊」　対象：数値　［例］＊＊＃＃

「＊」は必ず２つ必要です。数値の桁数が足りないときに，その余白部分を「＊」で埋めることができます。プリンタを使って打ち出したものを帳簿などに収めるときに，後で数字を書きかえられることがないようにしようというわけです。

6.「￥￥」　対象：数値

数値の前に￥(円マーク）をつけます。

7.「＊＊￥」　対象：数値

5と6を組み合わせたもので，¥をつけてさらに，足りない部分を＊で埋めます。

8．「^^^^」　対象：数値

数値を指定形式で表示します。小数点の位置を決めれば，それに応じた指数部をつけて表示します。

9．「！」　対象：文字列

文字列の最初の1文字だけを表示します。

10．「—」　対象：数値，文字列

「—」の後に書かれた書式指定子を指定子としてではなく文字として表示します。「#」を指定子としてではなく，記号として使いたいときなどに使います。

11．単位など

書式指定子に含まれていない文字や記号なら，そのまま画面に表示させることができます。X1には「年」「月」「日」などの9文字がグラフィック記号として用意されていますが，これを単位として表示させることもできます。また，スペースも単位の1つとして使うことができます。

●売上げ計算プログラム

このプログラムの160行では，11の方法で，数字の後に1文字をあけて「円」を表示させています。170行のINPUT文のすぐ後のヌルストリングが，その空白となります。また，ヌルストリングの後の「,」によって，データ入力をうながす「？」マークが画面に表示されるのを防いでいます。

ここまで7つの商品それぞれの売上げ数が入力されることになります。

データの入力がすべて正しく行なわれば，プログラムの中に特別な工夫をする必要はありませんが，実際にプログラムを使うことになると，入力の間違いやキーボードの押し間違いによって誤ったデータが入ってしまうことが，しばしば起こります。そのために，入力されたデータを画面で再確認させ，誤りがあればその時点で修正できるようにしたのが，LABEL “カクニン”の部分です。

まず，入力に使ったタイトルはそのままにして，FOR〜NEXTを使って品名，単価，数と表示しておきます。次に，「テイセイ　ハ　アリマスカ？」というメッセージを表示して確認を要求します。確認にはINKEY$を使います。YとNの大文字，小文字どちらでも正しい判断が行なわれるよう，INSTR関数を使い，この値によって比較しています。

INSTR関数というのは，文字列の中に含まれるある特定の文字列の位置をみつけてくれるものです。ちょっと横道にそれますが，たとえば次のプログラムを入力してください。

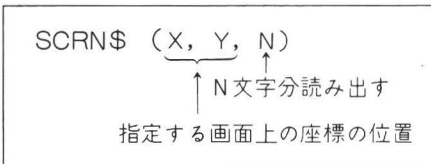
```
10 A$="レストラン エックスワン"
20 PRINT INSTR(A$,"エッ")
```

RUNをすると，7という数字が表示されます。これは，『レストラン　エックスワン』という文字列の頭から7番目の『エッ』という文字列がはじまりますよという意味になります。仮に20行をINSTR（A$“ウミ”）とすると，結果は「0」と表示されます。つまり，さがしている文字列が元の文字列にない場合には，関数の値は「0」というわけです。INSTRの書式は，

INSTR(開始位置, 文字列 1, 文字列 2)

となり，この20行の例のように開始位置を省略すると1からさがしはじめることになります。

レストランのプログラムに戻りましょう。

訂正がある場合，300行のLOCATE文とINPUT文によって，カーソルが1番目の品目の数の数値の上に表示されます。カーソルキーを使って，誤データを書きかえていきます。このときにカーソルのあったところの文字が変数DUMMY＄に入りますが，この変数はその名のとおりダミーで，特に使用されません。訂正されたデータは，310行からのFOR～NEXTによって，画面から読み込まれます。SCRN＄関数は，

という書き方をします。この関数は，X，Yで指定された画面上の座標位置（LOCATE文と同じ指定のしかたです）から，N文字分を読んで変数に代入する働きをします。ここでは，FOR～NEXTによって，27文字目から書かれた数字を次々に読み込んで，配列変数KAZ(I)に代入し直しています。したがって，変数KAZの中にはカーソルを動かして書きかえられた正しいデータが置きかえられるわけです。訂正がすべて終わると，290行の判断によってメインルーチンへ戻ります。

●プログラムで作表

350行から，表を書く部分です。このような表は，横に長くなりがちですから，80字モードとしています。130行で指定されていたCONSOLEは，この表示モードの切りかえによって解除されます。

まず，表のタイトルを横2倍文字で表示させ，次に，項目名の書き込まれる部分を表示させます。このような部分を，「**ヘッダー**」と呼びます。いわば表の見出しです。

たとえば会社の会計処理や在庫管理などの業務で，プリンタを使ってもっと長い表を作るようなときには，プリンタ用紙のページが改まるたびにこのヘッダーを書く必要がありますから，ヘッダーを書く部分だけをサブルーチンとしてまとめておく方法もよく使われます。また表の横の仕切線は，1つの種類しか使わないので，ここでは1つの文字列変数BO$に代入して繰り返し使っています。後は，FOR～NEXT文を使って，次々に項目を書いていきます。

このときに，LOCATEをまったく使わず，すべてTAB(n)を使っていますが，これは，プリンタを使って表の出力を行なうことを考えてのことです。つまり，プリンタに出力するときは，LOCATEの命令を使うことができませんから，縦方向は改行による送りだけに頼り，横方向はTAB(n)によって指定します。

440～530行のPRINTをLPRINTに直せば，プリンタで打ち出すことができるようになります。

●プリンタを使うと

表5-1は,プリンタを使って打ち出したものです。(実際は画面表示とは違って,縦の線がつながっていません)。これはプリンタで改行されるときの移動量が，画面表示よりも大きいために起こります。また,CSIZEによる文字サイズの変更はできません。

プリンタへ出力する場合，いろんなコントロールコードが用意されていますから，これを使うとよいでしょう。

〔プログラム5-2〕の中に示したのがプリンタコントロールコードです。

表5-1

ヒンメイ	カズ	タンカ(円)	キンガク
カレーライス	15	450	6,750
エビフライ	22	850	18,700
ハンバーグ	38	800	30,400
スパゲッティ	62	500	31,000
カニグラタン	30	960	28,800
ポークソテー	56	1,000	56,000
ミックスフライ	30	880	26,400
ゴウケイ			198,050

〔プログラム5-2〕

```
370 LOCATE0,0:CSIZE2:PRINT#0"ウリアゲヒョウ"
380 PRINT"┌────────┬───────┬───────┬──────────┐"
390 PRINT"| ヒ ン メ イ  |  カ  ズ | タンカ(円) |  キ ン ガ ク  |"
400 BO$="├────────┼───────┼───────┼──────────┤"
410 TTL=0
420 FOR I=1 TO 7
430 PRINTBO$
440 PRINT"|";USING"&           &";MENU$(I);
450 PRINTTAB(11);"|";TAB(14);USING"#,###";KAZ(I);
460 PRINTTAB(20);"|";TAB(23);USING"#,###";NEDAN(I);
470 PRINTTAB(29);"|";TAB(32);USING"#,###,###";KAZ(I)*NEDAN(I);
480 PRINTTAB(42);"|"
490 TTL=TTL+KAZ(I)*NEDAN(I)
500 NEXT
510 PRINT BO$
520 PRINT "| ゴ ウ ケ イ |          |          |";TAB(31);USING"##,###,
###";TTL;
530 PRINTTAB(42);"|":PRINT"└────────┴───────┴───────┴──────
────┘"
540 RETURN
550 DATA カレーライス,450,エビフライ,850,ハンバーグ,800,スパゲッティ,500,カニグラタン,
960,ポークソテー,1000,ミックスフライ,880
```

作表を行なう場合，TAB関数とPRINT USINGは密接な関係を持っています。TAB関数で決められた位置から桁数にかかわらず，スペースを加えて表示されるからです。

たとえば，「タンカ」の欄をもう少し幅を広くしたいときなどは，ヘッダーはもちろん，このプログラムでは450行以降のTAB関数で指定する値をすべて変えなければなりません。したがってこのような表を作るときには，先に方眼紙などでレイアウトを考えてからプログラム作りにとりかかった方が能率的です。また表を書きながら金額や総合計の計算を行なっている点にも注意してください。

なお，この〔プログラム5-2〕は後述のグラフ作成のところで使いますから，必ずカセットテープにSAVEしておいてください。

5-2 グラフを描く〔1〕～棒グラフ

●グラフの基本は

ビジネスに限らずあらゆる分野で，いろいろなデータを説明する方法として，グラフが使われます。たくさん並んだ数字よりもグラフの方が直接視覚にうったえますし，データの全体を把握することもたやすくなります。

X1の優れたグラフィック機能を利用して，グラフの描き方をみていきましょう。

グラフとひとことで言っても，さまざまな種類があります。まず頭に浮かぶのが棒グラフでしょう。ここでは身近な棒グラフを描くことからはじめることにします。

グラフを描くには，データが必要です（というよりもデータをグラフ化するというのが本来の考え方です)。まずデータの入力を行なう部分を作ります（プログラム5-3)。

〔プログラム5-3〕

```
100 LABEL"INPUT"
110 WIDTH 80:SCREEN0,0:CLS4
120 LOCATE0,0:INPUT"デ-タ スウ (20 マデ)",DA
130 IF DA<0 OR DA>20 THEN LOCATE0,0:PRINTCHR$(5):GOTO120
140 DIM DA(DA)
150 FOR I=1 TODA
160 PRINT "No.";I;:INPUT"",DA(I)
170 NEXT
180 RETURN
```

いくつのデータを扱うかを最初に入力して，この数によって配列宣言を行ないます。そして，おなじみのFOR～NEXTを使って配列変数の中にデータを入力していきます。

X1ではリターンキーだけを押すと，変数の内容はそれまでのままになります。これを利用して，データの値が0のときには，わざわざ0キーを押さないで済ませることもできます。

ここでは,最初に決めておかねばならないデータ数に対してDAという変数を使っています。さらに，配列変数としてDA（n）も使っています。普通の変数DAと配列変数DA（n）とはまったく別のものとして扱われるから心配することはありません。ここまでのプログラムを見てみましょう。

次に，キャラクタを使ってグラフを描いてみます〔プログラム5-4〕。

〔プログラム5-4〕

```
190 LABEL"GRAPH"
200 CLS
210 FOR I=1 TO DA
220   FOR J=1 TO DA(I)
230     PRINT"●";
240   NEXT:PRINT
250 NEXT
260 RETURN
```

FOR～NEXTを二重にして，内側のループでデータの数だけ「●」を横に並べていきます。セパレータとして“；”を使っているので“●”は間をあけずに並びます。240行のPRINT文は，1

つのデータについてのグラフ化が終わったら，改行させるためのものです。このPRINT文がないと「●」がすべてつながってしまうことになります。

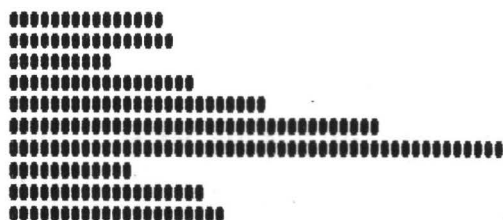
このような形のグラフを描くときに便利な関数にSTRING$があります。

STRING$(n, "文字")または、STRING$(n,数値)

と書きます。nで指定しておいた数だけ文字を書き並べます。数値をおいた場合はアスキーコードとみなして，対応する文字を書き並べます。これを使って作り変えたのが次の〔プログラム5-5〕です。

〔プログラム5-5〕

```
190 LABEL"GRAPH"
200 CLS
210 FOR I=1 TO DA
220   PRINT STRING$(DA(I),"●")
230 NEXT
240 RETURN
```

〔画面5-1〕

このようにすると，FOR～NEXT文を二重に使わなくても済みますから，プログラムが短くなりますし，プログラムの実行スピードも速くなります。

●目盛りをつける

このグラフですと，データの数値が小さいときはよいのですが，表示させる数が画面の横幅である80字を越えるとグラフが2行にわたってしまい，都合が悪くなります。また，データの単位が千や万の場合は，大変面倒な操作が必要になりますし，表示の精度も粗いものになってしまいます。グラフィックを使えば，横は640ドットを使うことができますから，かなり精密なグラフを描くことができます。次に，どのような単位のデータでもうまくグラフ表示を行なえるようにプログラムを工夫してみましょう。

「●」を並べて描いたグラフでは目盛りもついていませんでしたが，今度ははじめから目盛りや単位も考えてプログラムを作っていくことにします。目盛りをきれいに表示させるには，まず最大目盛り（フルスケール）をいくつにするかを決めなくてはなりません。普通，データの最大の値より大きい値で，きりのよいものをフルスケールの値にします。たとえば，データの最大の値が160ならフルスケールを200とし，800ならフルスケールを1000とします。このために，データの中でもっとも値の大きいものをさがしださなければなりません。この操作を行なうのが，サブルーチン“MAX”です。変数MAX（最初は0）と，配列に入っているデータを順次比較していき，MAXより大きいものがあればそれを変数MAXの中に収めます。最終的には，最大の値のデータが変数MAXに代入されることになります（プログラム5-6）。

〔プログラム5-6〕

```
190 LABEL"MAX"
200 FOR I=1 TO DA
210 IF DA(I)>MAX THEN MAX=DA(I)
220 NEXT
230 RETURN
```

このMAXの値がたとえば160なら，フルスケールを200とします。もしもっと数値が大きかったら，たとえばMAXの値が2800なら，フルスケールは3000になるわけです。MAXの数値の上

から2桁目を切り上げたものをフルスケールの値とすればよいわけです。この具体的な方法を考えてみましょう。まずMAXの値の単位がいくつになっているかを調べます。

たとえば，MAXが1600の場合，

$1600=1.6\times10^3$

と書き直せますから，単位は$10^3=1000$であることがわかります。このような形式を**指数形式**と呼び，1.6の部分を**仮数部**，10^3の部分を**指数部**と呼びます。上から2桁目を切り上げるには，この仮数部の数の整数部を取り出し，これに1を加えればよいわけです。この操作を行なっているのがサブルーチン"SCALE"の250行と260行です（プログラム5-7）。

〔プログラム5-7〕

```
240 LABEL"SCALE"
250 LM=LEN(STR$(MAX))-2
260 FUL=(INT(MAX/10^LM+1)*10^LM
270 SCAL=199/FUL
280 LINE(0,199)-(639,199),PSET,7
290 LINE(24,0)-(24,199)
300 STP=FUL¥(FUL*10^(-LM+1))
310 FOR I=0 TO FUL+1 STEP STP
320       IF I MOD STP*10=0 THEN GX=39 ELSE GX=29
330   LINE(24,199-I*SCAL)-(GX,199-I*SCAL),PSET,7
340 NEXT
350 RETURN
360 LABEL"GRAPH"
```

250行では指数部の値に相当するものを求めています。

符号桁
↓
␣1600
長さは5文字

␣1600
12

文字数から2を引いたもの（ここでは3）が単位のべき数になる。

$1600=1.6\times10^3$

260行では指数形式に直したMAXの数値の整数部に1を加え，さらに指数部をかけて実際のフルスケールとする値を求めています。

STR$は数値を文字列に変換する関数であることはもうご存じでしょう。INT関数は数値の値を越えない最大の整数を与える働きをします。数値が正の数であれば小数点以下を切り捨てることになりますし，負の数であれば小数点以下を切り上げることになります。またLENは文字列の長さを調べる関数です。

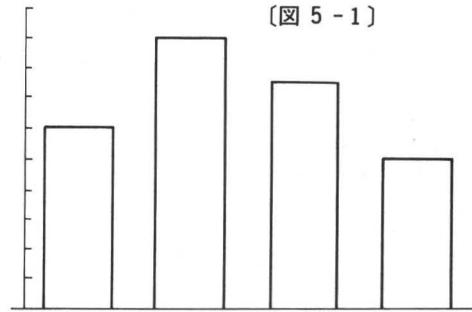
正の数の場合，符号桁の「+」が省略されていますから，文字数は見かけよりも1文字分多くなることに注意してください。MAXの値が1600なら，文字列に直したときの値は，符号桁の分を加えた5文字となります。

縦の棒グラフを考えた場合，横の棒グラフと異なって最大199ドットしか使えません。このドット数とフルスケールの値から目盛りのつけ方を決めていきます。フルスケールが2000の場合，199／2000≒0.1となりますから，1ドットが10を表わすことになります。これらを計算して，目盛りを描くのが，サブルーチン「SCALE」の270行からの部分です。300行の¥は整数の割算

の記号です。

いろいろ面倒な計算を行なっていますから，実際の数値をあてはめながら，どのような計算が行なわれているのか考えてみてください。

〔図 5-1〕

ここまでで，グラフの目盛りが描けました。今度は実際にこの目盛りにあるようにグラフを描く方法です。グラフの表示を図5-1のようにすることを考えます。データの数は，毎回入力するようにしてありましたから，この棒の幅もデータの数によって割り出し，画面内にバランスよく収まるようにします。画面の横幅は640ドットですが，左の方には目盛りがありますから実際にグラフを描くのはその右側の600ドット分とします。棒の左端の位置は，この600ドットをデータ数で割ったものとすれば，データがいくつになっても画面内にきちんと収まってきます。そして棒の幅もこの数値より小さくしておけば，隣りの棒に重なることもなくうまく表示できるはずです。これらを計算するのが，サブルーチン「GRAPH」の370行です（プログラム5-8）。

〔プログラム5-8〕

```
360 LABEL "GRAPH"
370 GSTEP=600/DA:GW=GSTEP*.8
380 FOR I=1 TO DA
390    LINE(-GSTEP+GSTEP*I+48,199)-(-GSTEP+GSTEP*I+GW+48,199-DA(I
)*SCAL),PSET,7,B
410 NEXT
420 RETURN
```

変数GSTEPに描きはじめのステップが入り，棒の幅はこれに0.8をかけたものとしました。データ数が多くなればGSTEPの値は小さくなり，GWの値も小さくなりますから，データ数にかかわらず，同じ比率でグラフを描くことができるようになります。

実際のグラフの棒は，グラフィックのLINE文のBモードで描くようにします。サブルーチン「SCALE」でデータの値の1に対してドット数がいくつになるかを求めて，この値を変数SCALEに代入しています。変数SCALEの値にそれぞれのデータをかけてやれば，実際の画面上の棒の高さを求めることができます。そして，下をそろえたグラフとするためには，画面の縦いっぱいの値である199からこの長さを引いたものがグラフの上端となるようにすればよいわけです。これらの処理を行なっているのが，380行からのFOR〜NEXTループです。

●グラフに色づけ

せっかくきれいなグラフが描けたのですから，色をつけることにしましょう。PAINT文を使えばよいのですが，基本色は黒をのぞくと7色しかありません。データが7つ以上の場合，同じ色を繰り返して使わなければなりません。変数Ｉに色コードを代入しておくことにすると，Ｉの値が，8のときは色コード1を指定し，9のときには色コード2を指定することにすればよいわけです。つまり，Ｉを7で割ったときのあまりの値を色コードとすればよいのです。このような場合には，MOD関数を使います（プログラム5-9）。

〔プログラム5-9〕

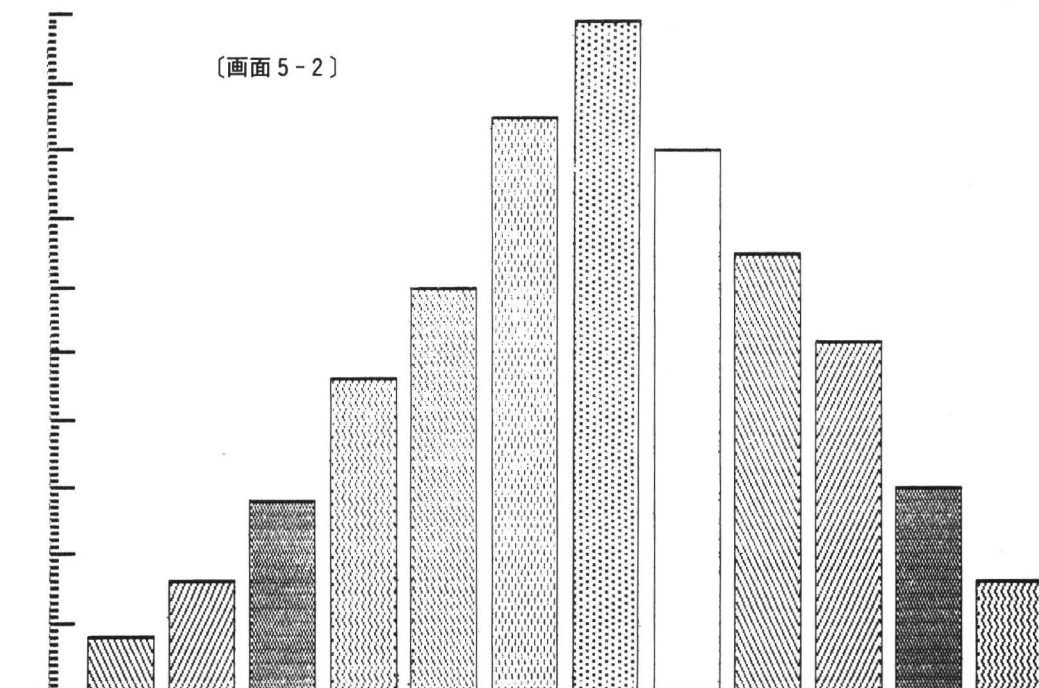
```
10 RESTORE 430
20 FOR I=1 TO 8:READ COL$,D$:COL$(I)=HEXCHR$(COL$)
30 NEXT
360 LABEL "GRAPH"
370 GSTEP=600/DA:GW=GSTEP*.8
380 FOR I=1 TO DA
390    LINE(-GSTEP+GSTEP*I+48,199)-(-GSTEP+GSTEP*I+GW+48,199-DA(I
)*SCAL),PSET,7,B
400    PAINT(-GSTEP+GSTEP*I+49,198),I MOD 7,7
410 NEXT
420 RETURN
```

PAINT文を使って色をつけると画面上で見る限り，大変きれいなグラフが描けます。しかし，プリンタを使って画面をそのまま紙の上に打ち出しても色は再現できません。プリンタを使うことを重点に考えれば，色をつけるよりも斜線やシマ模様にしておいた方がよいでしょう。

これにはタイリングペイントを使います。次のプログラムでは８つのパターンのデータを430行から用意しておき，プログラムの最初で配列変数COL$（8）に代入しています。これを，やはりMODを使う方法で順にペイントに使います。実際にプリンタでコピーした例もあわせてのせておきました。

```
430 DATA888888444444222222111111,2
440 DATA111111222222444444888888,3
450 DATAAAAAAA555555,1
460 DATA111111222222444444222222,4
470 DATA888888444444222222,5
480 DATA888888888888222222222222,6
490 DATAC0C0C00C0C0C,7
500 DATA888888888888555555222222222222555555,8
```

以上のサブルーチンをまとめて，きれいな棒グラフを描くようにしたのが次の〔プログラム5-10〕です。全体の流れをよく見てください。実行結果は画面５-２のようになります。

〔画面5-2〕

〔プログラム5-10〕

```
10 RESTORE 430
20 FOR I=1 TO 8:READ COL$,D$:COL$(I)=HEXCHR$(COL$)
30 NEXT
40 GOSUB"INPUT"
50 GOSUB"MAX"
60 GOSUB"SCALE"
70 GOSUB"GRAPH"
80 HCOPY0:END
100 LABEL"INPUT"
110 WIDTH 80:SCREEN0,0:CLS4
120 LOCATE0,0:INPUT"ﾃﾞｰﾀ ｽｳ (20 ﾏﾃﾞ)",DA
130 IF DA<0 OR DA>20 THEN LOCATE0,0:PRINTCHR$(5):GOTO120
140 DIM DA(DA)
150 FOR I=1 TO DA
160 PRINT "No.";I;:INPUT"",DA(I)
170 NEXT
180 RETURN
190 LABEL"MAX"
200 FOR I=1 TO DA
210 IF DA(I)>MAX THEN MAX=DA(I)
220 NEXT
230 RETURN
240 LABEL"SCALE"
250 LM=LEN(STR$(MAX))-2
260 FUL=(INT(MAX/10^LM+1))*10^LM
270 SCAL=199/FUL
280 LINE( 0,199)-(639,199),PSET,7
290 LINE(24,0)-(24,199)
300 STP=FUL¥(FUL*10^(-LM+1))
310 FOR I=0 TO FUL+1 STEP STP
320     IF I MOD STP*10=0 THEN GX=39 ELSE GX=29
330   LINE(24,199-I*SCAL)-(GX,199-I*SCAL),PSET,7
340 NEXT
350 RETURN
360 LABEL"GRAPH"
370 GSTEP=600/DA:GW=GSTEP*.8
380 FOR I=1 TO DA
390   LINE(-GSTEP+GSTEP*I+48,199)-(-GSTEP+GSTEP*I+GW+48,199-DA(I
)*SCAL),PSET,7,B
400 PAINT(-GSTEP+GSTEP*I+49,198),COL$(I MOD 8+1),7
410 NEXT
420 RETURN
430 DATA888888444444222222111111,2
440 DATA111111222222444444888888,3
450 DATAAAAAAA555555,1
460 DATA111111222222444444222222,4
470 DATA888888444444222222,5
480 DATA888888888888222222222222,6
490 DATAC0C0C00C0C0C,7
500 DATA888888888888555555222222222222555555,8
```

5-3 グラフを描く(2)～円グラフ

●円グラフを描く

それぞれのデータが占める割合を知りたいときに，非常に便利なのが「円グラフ」です。今度は，この円グラフを描いてみましょう。

X１に円グラフを描かせるのに必要な手順を考えてみましょう。

1．データを入力する。

2．データを大きい順に並べかえる。

3．すべてのデータの合計を計算し，各データが，合計に対してどの位の比率（構成比）になっているかを割り出す。

4．各データの構成比率から，円グラフにしたときの中心角を計算する。

5．画面にグラフィック表示する。

この一連の作業で円グラフが描けます。

まず，データの入力には，前に作った棒グラフのプログラムを一部変更して使うことができます。円グラフは，あまりたくさんのデータ数を扱うのには向きませんから，データの数を７までとします。また，今度はデータの項目名も入力できるようにしておきます。グラフ化したときに項目名もあわせて表示するようにしました。

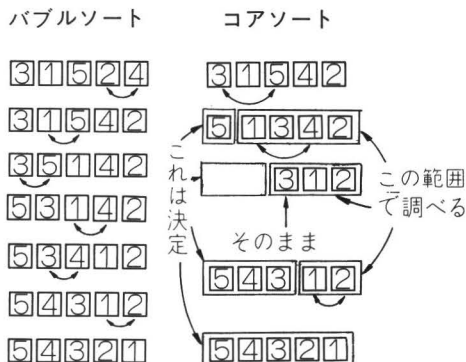
●ソートをすれば？

円グラフは構成比を表示するのに適していますから，一般には比率の大きい順に表示をします。ですから，構成比によるデータの並べかえ作業が必要になります。並べかえ作業を**ソート**（整列）といいます。ソートは，コンピュータに行なわせる仕事の代表的なものの１つです。人間が行なうのであれば，大変な時間がかかる作業ですが，コンピュータは極めて短い時間にこの作業を終えてくれます。とはいえ，データ数がとても多いときには，コンピュータといえども瞬時で仕事を終えるわけではありません。そのため作業の時間をちぢめるために，いくつかのソートの方法が考えられてきました。

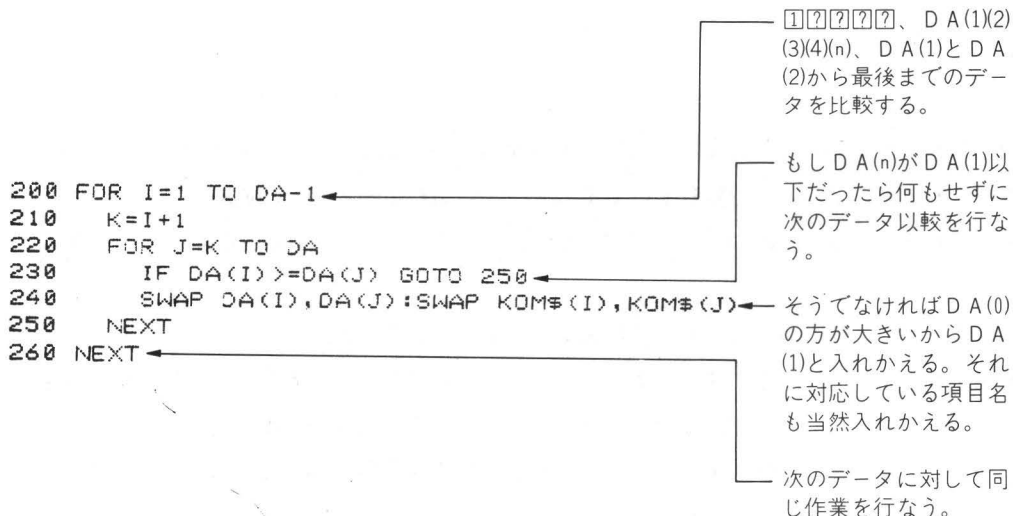
もっとも簡単な方法として，**バブルソート**（隣接交換法）と**コアソート**（逐次決定法）をあげることができます。**バブルソート**は，データの並びの隣り同士を比較して，後のものが前のものより大きい（小さい）ときはその２つを入れかえる，

〔図５－２〕

バブルソート　コアソート

31524　31542
31542　51342
35142　これは決定　312　この範囲で調べる
53142　そのまま
53412　54312
54312
54321　54321

ということを繰り返していく方法です。**コアソート**は，全体の中で一番大きい（小さい）データをさがしだし，それをまず一番最初に持ってきます。そして残りのデータの中の一番大きいものをまた前に持ってくる，という方法です。バブルソートとコアソートの違いは，図5-2を見ていただければわかることと思います。

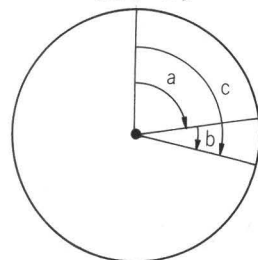
ここではコアソートを利用します。プログラムでは，サブルーチン「SORT」の部分です。プログラムに即して見ていきましょう。

I＝1のとき，

以上の作業によって，入力されたデータは，大きい順に整理されます。

●構成比率の計算

次に，グラフを描くための下準備も兼ねて，データの構成比率の計算をします。サブルーチン「STEP」がこのための部分です。変数GKEIに代入されているデータの総合計で個々のデータを割って比率を計算し，PER（n）という配列変数に代入しておきます。

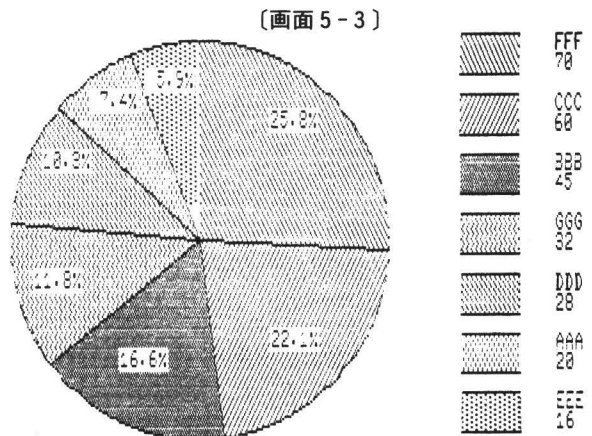
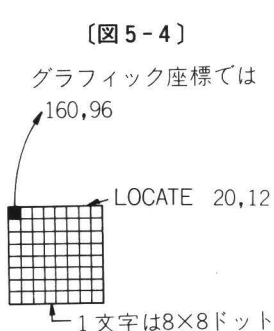
これで，円グラフを作るために必要な下準備が整いました。円グラフにするための中心角は，360度にPER(I)を掛けたものになります。円グラフの本体を描くルーチンが，「円GRAPH」です。外形の円は，CIRCLE文によって描いています。あとは，各データの比率から，中心角を決めて半径の線を描き，色を塗ってやればよいわけです。中心角がわかれば，前に使った円の公式によって，円周上の座標を計算することができます。ただし，2番目以降のデータについては，それ以前のデータの持つ中心角にそのデータの中心角を加えたときの座標を求める必要があります（図5-3）。

〔図5-3〕

a＝1番目のデータの中心角
b＝2番目のデータの中心角
c＝a＋b
実際に2番目のデータの区切りの線を描く角度

また，円を描く公式では，0度の位置は画面上で右の方向で，そこから右回りに角度が増していきますから，スタートの位置を上にするための補正も加えなければなりません。これらの計算を行なうのが400～440行です。

ここではPAIという関数が使われています。PAI (x) とすると，円周率のX倍の値が得られます。たとえば円の円周を求める式は，S＝2πr（r：半径）ですが，これをS=PAI（2）＊Rと表わすことができます。グラフィックキャラクタのπも円周率として使うことができます。PAI（1)＝πというわけです。πの場合，後に＃をつけてπ＃として使うと，3.141592653589798として使うことができます。BASICでは角度の単位はラジアンですから，PAI(2)＝2πは角度で360度になります。したがってここでは，角度をラジアンに変換することなく計算されています。各データの中心角の大きさは，PAI（2）に構成比率RER（n）を掛けたものになります。そして実際に必要な0の点からの角度は，これにそれ以前のデータを加えたものとして計算し，配列変数STP（n）に代入されます。STP（0）には－PAI（1)/2という値が代入してありますが，これは角度で－90度となります。1番目のデータに対する座標は，円の真上の位置である－90度を基点として計算されます。また，PAINT文によって色を塗りはじめる位置は，その扇形の中心角の中央になるよう計算しています。各データの区切り線を描く位置は，GX, GYに，PAINT文の塗りはじめの位置はPX（n），PY（n）に代入され，450行で線を引き，460行で色を塗っています。

次に，それぞれの構成比をグラフに重ねて表示させます。また，円グラフを項目に従って色を塗りわけます。これをサブルーチン「ハンレイ」で行なっています。書き込む位置は，扇形の中央あたりにしたいので，PAINT文の塗りはじめの位置として計算してある，PX(n)，PY(n)を利用します。ここでは，画面を80字モードで使っていますから，文字は80文字×25行，グラフィックは640×200ドットになっています。1文字は8×8ドットですから，縦，横とも，グラフィック座標を8で割ったものが，文字の表示位置となります（図5－4)。ドット表示での座標位置を示すPX（n），PY（n）を8で割った値を整数化した値をLOCATE文のパラメータとして使用すればよいわけです。こうして表示される文字のバックを黒にすると文字も読みやすくなります。この文字の位置を示す変数MLOCX, MLOCYの値を8倍してもう一度グラフィック座標に戻してからLINE文のBFモードを使用しています。

〔図5－4〕

〔画面5－3〕

また，画面の右側には小さな長方形を並べて，円グラフを塗りわけた色と同じ色を塗っておきます。この隣りに項目名とデータの数値を表示しました。

棒グラフのときと同じように，プリンタによってコピーを取るには，タイリングペイントによって模様塗りを行なった方がよいことはいうまでもありません。

このプログラムは汎用性がありますから，カセットテープに記録して保存しておくとよいでしょう（プログラム５-11)。実行結果は画面５-３のようになります。

〔プログラム５-11〕

```
10 INIT
20 GOSUB "INPUT"
30 GOSUB "SORT"
40 GOSUB "STEP"
50 GOSUB "円GRAPH"
60 GOSUB "ﾊﾝﾚｲ"
70 END
80 LABEL "INPUT"
90 WIDTH 80:SCREEN 0,0:CLS 4
100 LOCATE 0,0:INPUT "ﾃﾞｰﾀ ｽｳ (7 ﾏﾃﾞ)",DA
110 IF DA<0 OR DA>7 THEN LOCATE 0,0:PRINT CHR$(5):GOTO 100
120 DIM DA(DA)
130 FOR I=1 TO DA
140   LOCATE 0,I:PRINT "No.";I;:INPUT "ｺｳﾓｸ ﾒｲ ",KOM$(I)
150   LOCATE 20,I:INPUT "ｶｽﾞ ",DA(I)
160   GKEI=GKEI+DA(I)
170 NEXT
180 RETURN
190 LABEL "SORT"
200 FOR I=1 TO DA-1
210   K=I+1
220   FOR J=K TO DA
230     IF DA(I)>=DA(J) GOTO 250
240     SWAP DA(I),DA(J):SWAP KOM$(I),KOM$(J)
250   NEXT
260 NEXT
270 RETURN
280 LABEL "STEP"
290 FOR I=1 TO DA
300   PER(I)=DA(I)/GKEI
310 NEXT
```

らんだむめも

FRE・SIZE

コンピュータのメモリには，決まった容量というのがあります。キーボードからプログラムを打ち込んでいったり，変数にデータを入れていくと，それらはメモリの中に収められていきますから，当然メモリの残り容量はしだいに減っていきます．

現在どの位のメモリが空(アキ)として残っているかを調べる関数がFRE(n)，またはSIZEです．この２つは，まったく同じ命令です．FRE関数の引数はダミーですから何を入れてもさしつかえありません．

プログラムを打ち込んでいて，残りメモリが気になったときはいつでもこの関数によって調べることができます．たとえば，PRINT FRE(0)として，8200と表示されたとすると，あと約8200文字くらいまでは入れられるということになります．ただし，これはプログラム自体だけの話で，変数やデータなどで使う分を考えてはいません．特に注意しなくてはならないのは配列変数を使う場合です．プログラム中で配列宣言が行なわれると，その配列のための領域をメモリの中に確保しますから，プログラムを入力し終わったときの残りメモリが少ない場合，配列宣言が実行されたとたんにOut of memoryになることがあります．数値変数の場合，１つの配列要素について内部表現によって５バイト（単精度の場合）が必要ですから，たとえば，DIM A(255)とした場合，256×５＝1280バイトも使うことになります．プログラムを作るときには，どうしても必要なものだけ配列を使い，保存の必要のない計算の途中経過などは，普通の変数を使うようにした方がよいでしょう．

```
320 RETURN
330 LABEL "円GRAPH"
340 CLS 4
350 CIRCLE(180,100),80,7
360 LINE(180,100)-(180,20),PSET,7
370 STP(0)=-PAI(1)/2
380 FOR I=1 TO DA
390   STP(I)=PAI(2)*PER(I)+STP(I-1):CEN=PAI(2)*PER(I)/2+STP(I-1)
400   GX=COS(STP(I))*160+180:GY=SIN(STP(I))*80+100
410   PX(I)=COS(CEN)*120:PY(I)=SIN(CEN)*60
420   LINE(180,100)-(GX,GY),PSET,7
430   PAINT(PX(I)+180,PY(I)+100),(I-1) MOD 7+1,7
440 NEXT
450 RETURN
460 LABEL "ﾊﾝﾚｲ"
470 FOR I=1 TO DA
480   MLOCX=INT(20+PX(I)/8):MLOCY=INT(12+PY(I)/8)
490   LOCATE MLOCX,MLOCY:PRINT USING "##.#%";PER(I)*100
500   LINE(MLOCX*8-1,MLOCY*8-1)-(MLOCX*8+40,MLOCY*8+8),PSET,0,BF
510   WX=(I-1)*24
520   LINE(400,WX)-(450,WX+16),PSET,7,B
530   PAINT(401,WX+1),(I-1) MOD 7+1,7
540   LOCATE 60,(I-1)*3:PRINT KOM$(I):LOCATE 59,(I-1)*3+1:PRINT
DA(I)
550 NEXT
560 RETURN
```

5-4 プログラムのドッキング

●ドッキングするには？

データの作表からはじまって，棒グラフ・円グラフを描くまで，3本のプログラムを作ってきました。これらは，入力されたデータの形を変えて表示させる方法といえます。データを別の角度から見直す方法といってもよいでしょう。ここでは，前に作った作表のプログラム（プログラム5-1）とグラフを作るプログラムとを組み合わせて，グラフと表とを一緒に表示させてみましょう。ビジネスでは，データを別の角度から見直したり，まとめて確認したりという作業が必要となるからです。

作表のサンプルは前に作ったレストランの売上げ集計のプログラムを使います。どの商品の売れ行きがよいかの傾向を見るために，ここでは円グラフを使うことにします。

まず，作表のプログラムをテープからロードします。このプログラムに，グラフを描く部分をつけ足せばよいわけです。ところが，後から円グラフのプログラムをLOADすると，それまで入っていた作表のプログラムは消えてしまうことになります。このような場合，MERGE（マージ）という方法で2本のプログラムを結合します。MERGEを行なうためには，後から読み込むプログラムが**アスキー形式**でSAVEされていなくてはいけません。通常のセーブで，X1はプログラムの内容を2進数の信号に変換して記録しています。このようなセーブ方式を**バイナリー形式**と呼びます。

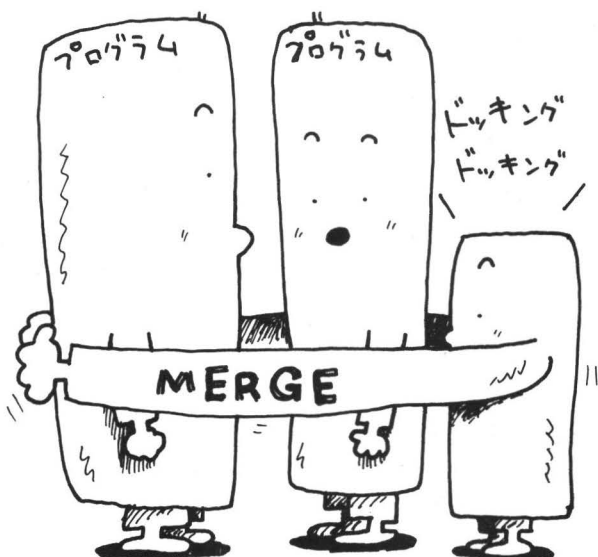
アスキー形式というのは，打ち込んだ文字をそのままの形でセーブするものです。この形式でセーブを行なうと，バイナリー形式でセーブするのに比べて，数倍の時間がかかります。

とりあえず，先ほどLOADした作表のプログラムを改めてアスキーセーブしましょう。別のカセットテープを用意してください。ファイルの名前は「サクヒョウ」としましょう。

```
SAVE"サクヒョウ", A
```

とすれば，このプログラムのアスキーセーブが行なわれます。ファイルネームの後にカンマで区切ってAをつけると，アスキーセーブされることになります。アスキー形式も，プログラムの読み込みにはLOAD文を使います。

アスキーセーブを行なうと，カセットデッキが何回か動作と停止を繰り返します。これは，256文字分ずつにまとめてプログラムの読み込みを行なっているためです。

作表プログラムのアスキーセーブが

終了したら，今度は円グラフのプログラム〔プログラム5-11〕をロードします。このプログラムのうち，データを入力する部分は，作表プログラムにもありますから不要です。

プログラムの不要な部分を消すには，DELETE文を使います。DELETEのパラメーターは，LISTとほぼ同じですが，改めてここで確かめてみましょう（表5-2)。パラメータを省略すると，作動しません。

表5－2

LIST	全プログラムのリストを表示	DELETE	全プログラムを削除
LISTm—	m行から後の全プログラムのリストを表示	DELETEm—	m行から後の全プログラムを削除
LISTm—m	m行からn行までのプログラムのリストを表示	DELETEm—n	m行からn行までのプログラムを削除
LIST—n	プログラムの最初からn行までのリストを表示	DELETE—n	プログラムの最初からn行までを削除

ハイフンのかわりに「,」を使うこともできます。

この場合，100から180行を削除するわけですから，

```
DELETE  100, 180
```

とします。プログラムのこの部分がきちんと削除されたかどうかをリストをしてみて確認しておいてください。

100から180行の他に，10行と40行も削除します。さらに，作表のプログラムからこの円グラフのプログラムを1つのサブルーチンとして呼び出して使うために，10行にLABELをつけてしまいます。

```
10  LABEL "円グラフ"
```

そして，90行のENDをRETURNに直しておきます。これで，円グラフを描くプログラムの全体が「円グラフ」という名前の大きなサブルーチンの形になりました。

次に作表のプログラムをMERGEすればよいのですが，このままでは行番号の重なる部分ができてしまいます。アスキーセーブされたプログラムをロード，またはマージすると，セーブされていたものがキーボードから打ち込まれたのと同じことになりますから，このままではうまくいきません。そのために，円グラフを描くサブルーチンの行番号を変えておかなければなりません。作表のプログラムは590行で終わっていましたから，円グラフを描くサブルーチンを600行からはじまるように直します。このためには，RENUM（リナンバー）というコマンドを使います。

●リナンバーの技術

このRENUMというコマンドは，

と書きます。mからの行番号をnからに書き直す命令で，書き直されるプログラムのステップを st のところで指定することもできます。st を省略すると行番号は10ずつになり，mを省略すると，プログラム全体の行番号が書きかえられます。パラメータをすべて省略すると，10からはじまる10ステップの行番号になります。ですから，プログラムの手直しをしていって行番号がつまってしまって困ったときなど，RENUMによって間隔を整えることができます。

たとえば，

```
 ┌──────────────新しい行番号
 │     ┌────────古い行番号
 ↓     ↓
RENUM 600, 500
```

とすると，500行からの部分が600行へ移動し，500から590までの10行に別の部分を書き込むことができるというわけです。このような場合、新しい行番号は前の行番号より大きくないと当然エラーになります。新しい行番号だけを指定して全体を変化させるときは小さくてもかまいません。

RENUMBERを行なうと，GOTO, GOSUB, RESTOREなどで指定する行番号もすべて書きかえられます。そのときにもし，まだ書かれていない行番号へのGOTO命令などがあると，そこは65535という番号になります。

●もういちど，プログラムをつなげる

ここでは，作表のプログラムが590行で終わってしまいましたから，

RENUM 600

として600行からはじまるようにして直しておきます。

次に，アスキーセーブされた作表のカセットをセットし，MERGEと命令します。セーブのときと同じように，テープが動作・停止を何回か繰り返して止まります。リストをとってみると，作表のプログラムと円グラフのプログラムが両方とも間違いなく書き込まれていることがわかります。後は，作表のプログラムを少し手直しして，大きなサブルーチン「円グラフ」を呼び出すようにすればよいのです。

MERGEした後で変更，追加する部分は次のとおりです。

```
10 WIDTH80:SCREEN0,0:CLS4
20 GOSUB"デ゛ータ"
30 GOSUB"カズ゛"
40 GOSUB"カクニン"
50 GOSUB"ヒョウ"
60 GOSUB"円ク゛ラフ"
65 END
```

●X１ではこんなことも！

MERGEを行なうのに，いちいちカセットテープを使うのでは時間もかかって面倒だという人のために，X１ならではの便利な方法を紹介しましょう。それは，グラフィックVRAMを外部メモリとして使う方法です。

X１のグラフィックVRAM（以下GRAMとします）は，48k BYTEの容量を持っています。このGRAMをグラフィック専用のRAMとしてではなく，普通のメモリとしてプログラムやデータを書き込んでしまうことが，X１ではできるのです。

このようなGRAMの使用目的を決めるのが，

OPTION SCREEN n

というステートメントです。nの値が１のときは，GRAMはこれまでどおりグラフィックを描くために使用されます。nを２にすると，外部メモリとして使うことができるようになります。画面５－４のように表示されたら，GRAMの内容をクリアする意味も含めてイニシャライズ（初期化）を行ないます。

〔画面５－４〕

```
OPTION SCREEN 2
```

INIT "MEM : "

としてリターンキーを押してください。

Are you sure? (y or n)

と確認を求めてきます。よければ，Ｙキーを押します。yとnは大文字でも小文字でもかまいません。

"MEM : "は，GRAMを示すファイルディスクリプタと呼ばれるものです。ファイルディスクリプタは，そのコマンドを働かせる対象の周辺機器を指定するものです。ファイルディスクリプタは次のようなものがあります。

SCR：画面（コントロールコードを実行する）

CRT：画面（コントロールコードを表示する）

KEY：キーボード

LPT：プリンタ

CAS：カセットテープ

MEM：グラフィックVRAM

EMM０～９：外部メモリ

これらのファイルディスクリプタの使い方は少々面倒なのですが，便利な方法もありますから，１つだけ覚えておいてください。プログラムのカセットテープをセットし，

RUN "CAS :

とすると，自動的にテープからの読み込みが行なわれ，終わるとすぐプログラムが走りはじめます。

さて，今までテープを使ってきたのと同じ手順をGRAMを使って行なってみましょう。

まず，作表のプログラムをテープから読み込みます。そしてこれをGRAMにアスキー形式でセーブしておきます。このときの画面は画面５－５のようになります。

次に円グラフのプログラムをLOADし，先ほどと同じようにリナンバーします。そして，GRAMにセーブしてある作表のプログラムをMERGEします。

便利な方法として，FILESを使ってみましょう。

FILES "MEM：

と入力してください。画面には，

Asc "MEMO：サクヒョウ…

〔画面5－5〕

```
LOAD"CAS:
OK
SAVE"MEM:ｻｸﾋｮｳ",A
OK
```

と表示されます。カーソルをAscのAのところまで移動させ，MERGEと打ち込んでリターンキーを押します。すぐにOKが表示されますが，これでOKなのです。リストをとって確認してみてください。本当にプログラムのMERGEが完了しています。

この方法を使うと，カセットテープに比べて大変手早くすませることができます。SAVE，LOADも含めて，ファイルディスクリプタを指定するだけですから，使い方も簡単です。

ただし，ファイルディスクリプタはすべて大文字を使わないとエラーになります。

ファイルディスクリプタを十分に利用できるようになれば，あなたも一人前のX1のプログラマです。

5-5 メニューを使えば

●プログラムの組み立て方

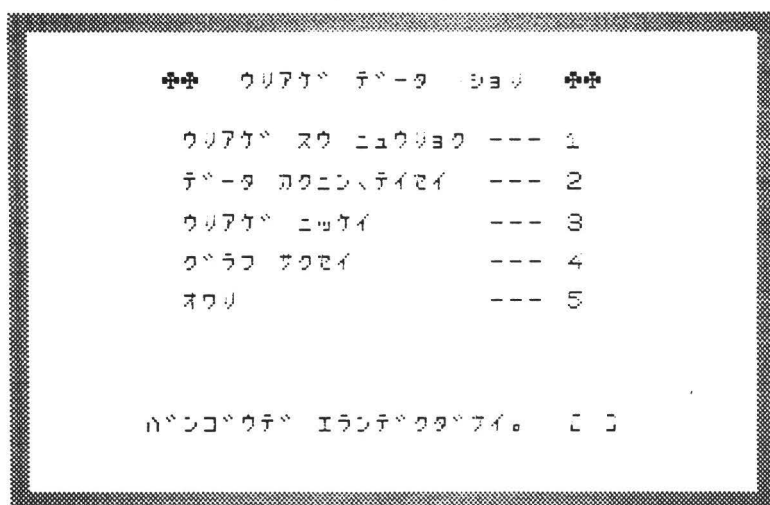
さてみなさんは，作表とグラフのプログラムをつなげて，１本のものにまとめあげ，さっそく利用している人もいることでしょう。もちろん，このままでも十分に使えるプログラムですが，プログラムの流れがはじめから終わりまで１つで，つまりは１回限りの処理しかできません。しかし，ある商品の売上げ数が増えたら全体の売上げは，どう変化するかといったことがわかれば，翌日の仕事の目安を立てることもできます。

このプログラムは，全体が５つのサブルーチンの集まりで構成されています。必要に応じて希望のサブルーチンを自由に呼べるようにしておくと,非常に実用的なプログラムになります。次の画面５－６が表示されて，必要な処理を選択できるようであれば，理想的といえます。

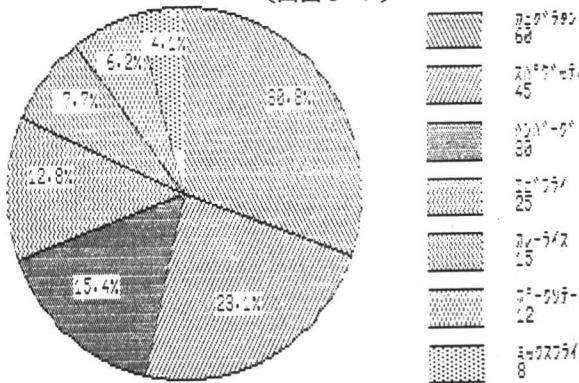
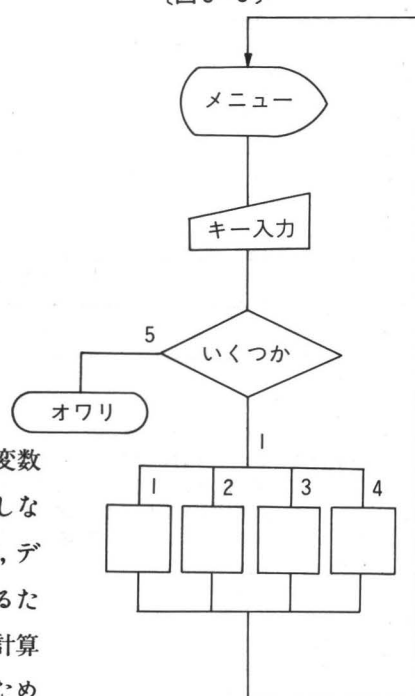
〔画面５－６〕

```
♣♣   ウリアゲ データ ショリ   ♣♣

  ウリアゲ スウ ニュウリョク ---  1
  データ カクニン、テイセイ  ---  2
  ウリアゲ ニッケイ          ---  3
  グラフ サクセイ            ---  4
  オワリ                     ---  5


バンゴウデ エランデクダサイ。   [ ]
```

本でいえば，目次にあたるこのような形式を一般に「**メニュー**」と呼びます。**メニュー形式**は，ビジネス用のプログラムには必ずといってよいほどよく使われています。

「５」で指定する「オワリ」以外の４つの仕事を実行し終わったら，必ずこのメニューを表示するところに戻ってくるように，操作もしやすく，誰にでも手軽に扱えるプログラムになります。このようなプログラムにするための考え方を図にすると，図５－５のようになります。全体のリストは〔プログラム５-12〕です（画面５－７）。前の〔プログラム５－１〕の70行にあったラベル「データ」の部分は，リナンバーによって100行から移り，その間に新たにメニューを表示する部分とキー入力（選択）の部分を入れてあります。

しかし，前の〔プログラム５－１〕をリナンバーしてこの部分をつけ加えるだけでは，いろいろ不都合な点が出てきます。つまり，グラフがうまく描けなくなったり，画面表示が重なってしまったり，また，表などを見る間もないうちに，メニュー画面に戻ってしまうというようなことです。

〔画面５－７〕　〔図５－５〕

メニュー形式を取り入れるためには，全体を通しての変数の動きや画面の表示モード，表示方法について，再検討しなくてはなりません。ある数の合計を計算する部分などは，データの変更があることを考えて，そのルーチンが呼ばれるたびに，変数の内容を一度完全にクリアしてからもう一度計算しないと不都合が生じます。また，見やすい画面作りのために表示モードの切りかえを行なうのも，各サブルーチンの最初で行なうようにしておくことも必要です。

〔プログラム５-12〕

```
10 GOSUB"ﾃﾞｰﾀ"
20 WIDTH40:SCREEN0,0:CLS4
30 COLOR4:LINE(0,0)-(39,24),"▓",B
40 LOCATE 8,3:PRINT"♣♣  ｳﾘｱｹﾞ ﾃﾞｰﾀ ｼｮﾘ  ♣♣"
50 LOCATE 9,6:COLOR6:PRINT"ｳﾘｱｹﾞ ｽｳ ﾆｭｳﾘｮｸ --- 1"
55 LOCATE 9,8:COLOR3:PRINT"ﾃﾞｰﾀ ｶｸﾆﾝ､ﾃｲｾｲ  --- 2"
60 LOCATE 9,10:COLOR5:PRINT"ｳﾘｱｹﾞ ﾆｯｹｲ       --- 3"
65 LOCATE 9,12:COLOR6:PRINT"ｸﾞﾗﾌ ｻｸｾｲ        --- 4"
70 LOCATE 9,14:COLOR3:PRINT"ｵﾜﾘ             --- 5"
75 LOCATE7,20:COLOR5:PRINT"ﾊﾞﾝｺﾞｳﾃﾞ ｴﾗﾝﾃﾞｸﾀﾞｻｲ｡   [ ]"
80 LOCATE30,20:REPEAT:Z$=INKEY$(1):UNTIL VAL(Z$)>0 AND VAL(Z$)<6
85 IF Z$="5" THEN END
90 ON VAL(Z$) GOSUB"ｶｽﾞ","ｶｸﾆﾝ","ﾋｮｳ","円ｸﾞﾗﾌ"
95 GOTO 20
100 LABEL"ﾃﾞｰﾀ"
110 RESTORE590:FOR I=1 TO 7
120 READ MENU$(I),NEDAN(I):NEXT
130 RETURN
140 LABEL"ｶｽﾞ":CLS:COLOR7
150 PRINT "ﾋﾝﾒｲ              ﾀﾝｶ           ｶｽﾞ"
160 PRINTSTRING$(30,"─"):CONSOLE2,23
170 FOR I=1 TO 7
180 LOCATE0,I+1:PRINT USING"&           &";MENU$(I);
190 PRINT USING"###,### 円";NEDAN(I),
200 LOCATE28,I+1:INPUT"",KAZ(I)
220 NEXT
230 RETURN
240 LABEL"ｶｸﾆﾝ":WIDTH40:CLS:COLOR7
250 FOR I=1 TO 7
260 LOCATE0,I+1:PRINT USING"&           &";MENU$(I);
270 PRINT USING"###,### 円";NEDAN(I),
280 LOCATE27,I+1:PRINT KAZ(I)
290 NEXT
```

```
300 LOCATE0,12:PRINT "ﾃｲｾｲ ﾊ ｱﾘﾏｽｶ? (Y/N) ";
310 YN$="YyNn"
320 REPEAT:A$=INKEY$(1):AN=INSTR(YN$,A$):UNTIL AN<>0
330 IF AN>2 THEN RETURN
340 LOCATE28,2:INPUT"", DUMMY$
350 FOR Y=2 TO 8
360 KAZ(Y-1)=VAL(SCRN$(27,Y,5))
370 NEXT
380 GOTO 250
390 LABEL"ﾋｮｳ"
400 WIDTH80:CLS 4:COLOR7
410 LOCATE0,0:CSIZE2:PRINT#0"ｳﾘｱｹﾞﾋｮｳ":CSIZE
420 PRINT"┌──────────┬─────────┬─────────┬─────────────┐"
430 PRINT"│ ﾋ ﾝ ﾒ ｲ  │  ｶ  ｽﾞ  │ ﾀﾝｶ(円) │   ｷ ﾝ ｶﾞ ｸ   │"
440 BO$="├──────────┼─────────┼─────────┼─────────────┤"
450 TTL=0
460 FOR I=1 TO 7
470 PRINTBO$
480 PRINT"│";USING"&          &";MENU$(I);
490 PRINTTAB(11);"│";TAB(14);USING"#,###";KAZ(I);
500 PRINTTAB(20);"│";TAB(23);USING"#,###";NEDAN(I);
510 PRINTTAB(29);"│";TAB(32);USING"#,###,###";KAZ(I)*NEDAN(I);
520 PRINTTAB(42);"│"
530 TTL=TTL+KAZ(I)*NEDAN(I)
540 NEXT
550 PRINT BO$
560 PRINT "│ ｺﾞ ｳ ｹ ｲ │         │         │";TAB(31);USING"##,###,
###";TTL;
570 PRINTTAB(42);"│":PRINT"└──────────┴─────────┴─────────┴─────────
─────┘"
575 LOCATE0,23:PRINT"OK --- ｽﾍﾟｰｽ"
580 Z$=INKEY$:IF Z$=CHR$(32) THEN RETURN ELSE 580
590 DATA ｶﾚｰﾗｲｽ,450,ｴﾋﾞﾌﾗｲ,850,ﾊﾝﾊﾞｰｸﾞ,800,ｽﾊﾟｹﾞｯﾃｨ,500,ｶﾆｸﾞﾗﾀﾝ,
960,ﾎﾟｰｸｿﾃｰ,1000,ﾐｯｸｽﾌﾗｲ,880
600 LABEL"円ｸﾞﾗﾌ":WIDTH80:CLS4:COLOR7
610 GKEI=0:DA=7:FOR I=1 TO 7:DA(I)=KAZ(I):KOM$(I)=MENU$(I):GKEI=
GKEI+DA(I):NEXT
620 GOSUB"SOART"
630 GOSUB"STEP"
640 GOSUB"円GRAPH"
650 GOSUB"ﾊﾝﾚｲ"
655 LOCATE0,22:PRINT"OK --- ｽﾍﾟｰｽ"
660 Z$=INKEY$:IF Z$=CHR$(32) THEN RETURN ELSE 660
670 LABEL"SOART"
680 FOR I=1 TO DA-1
690   K=I+1
700   FOR J=K TO DA
710     IF DA(I)>=DA(J) GOTO 730
720       SWAP DA(I),DA(J):SWAP KOM$(I),KOM$(J)
730   NEXT
740 NEXT
750 RETURN
760 LABEL"STEP"
770 FOR I=1 TO DA
780   PER(I)=DA(I)/GKEI
790 NEXT
800 RETURN
810 LABEL"円GRAPH"
820 CLS4
830 CIRCLE(180,100),80,7
840 LINE(180,100)-(180,20),PSET,7
850 STP(0)=-PAI(1)/2
860 FOR I=1 TO DA
870   STP(I)=PAI(2)*PER(I)+STP(I-1):CEN=PAI(2)*PER(I)/2+STP(I-1)
880   GX=COS(STP(I))*160+180:GY=SIN(STP(I))*80+100
890   PX(I)=COS(CEN)*120:PY(I)=SIN(CEN)*60
```


10100000

A0

```
900    LINE(180,100)-(GX,GY),PSET,7
910    PAINT(PX(I)+180,PY(I)+100),(I-1) MOD 7+1,7
920 NEXT
930 RETURN
940 LABEL"ﾊﾝﾚｲ"
950 FOR I=1 TO DA
960    MLOCX=INT(20+PX(I)/8):MLOCY=INT(12+PY(I)/8)
970    LOCATE MLOCX,MLOCY:PRINT USING"##.#%";PER(I)*100
980    LINE(MLOCX*8-1,MLOCY*8-1)-(MLOCX*8+40,MLOCY*8+8),PSET,0,BF
990    WX=(I-1)*24
1000    LINE(400,WX)-(450,WX+16),PSET,7,B
1010    PAINT(401,WX+1),(I-1) MOD 7+1,7
1020    LOCATE60,(I-1)*3:PRINTKOM$(I):LOCATE59,(I-1)*3+1:PRINT DA
(I)
1030 NEXT
1040 RETURN
```

●メニューからサブルーチンを呼び出すには

メニューから各サブルーチンを呼び出すには，メニューにつけられている番号によって行ない，ON～GOSUB文によって振り分けます。キー入力の処理方法は，これまでにもしばしば使ってきたREPEAT～UNTILLとINKEY$関数を組み合わせた方法で，１つのキーで選択の入力ができます。

各サブルーチンは，これまでに作った基本のものとほとんど同じですが，画面モードの変更などをうまく行なうように各サブルーチンのはじめの部分に，いくつかの命令を追加しました。このような後処理を行なうことを，「パッチあて」とか「パッチ処理」と呼びます。「パッチ」というのは，「つぎあて」のことです。プログラムにつぎあてをする，というわけです。もちろん，ここではプログラム作りの練習ですから，こういった後処理も行なわなければなりませんが，プログラムを作るときには，あらかじめきちんと設計をして，パッチ処理の必要のないよう心がけると良いでしょう。パッチ処理の多いプログラムは，どうしてもエラーが起きやすくなり，バグの発見の能率が悪くなります。

このプログラムは，項目を増やすなどの手を加えると，簡単なデータ処理などに使うことができます。各サブルーチンの働きや変数の使い方をよく確かめておいてください。

5-6 プリンタを使えば

●プリンタのいろいろ

作表グラフのプログラムで，プリンタを使うことを少し考えてみました。みなさんの中にはすでにプリンタを活用している人もいることでしょう。Ｘ１の周辺機器の中ではもっとも一般的でしかも非常に便利なものです。

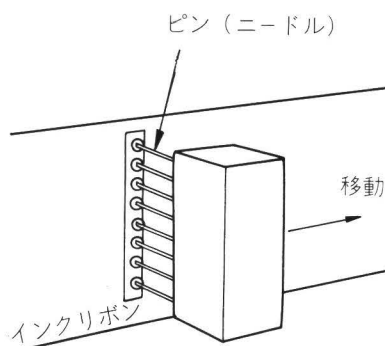
ここではプリンタの使い方を少し詳しく見ていきましょう。現在Ｘ１のシステムに用意されているのは，CZ-800Ｐという名のプリンタです。プリンタは，もうご存じのように紙にコンピュータから送り出される文字を打ち出す（印字するといいます）機器です。印字の方式には，ドットインパクト式，放電式，感熱式，インクジェット式などの種類があります。

放電式というのは，アルミ箔の上に薄い紙を重ねた用紙を使い，電気放電（スパーク）によって表面の紙を破壊し，アルミ箔を露出させて文字を形づくるものです。音が静かであるという長所はありますが，文字が読みにくいという欠点もあります。

感熱式は，別名サーマルプリンタと呼ばれるもので，熱に反応する特殊な用紙を用いるものです。このタイプも音が静かなところから，最近では一部のワードプロセッサにも採用されています。

〔図５－６〕

インクジェット式は，非常に細いノズルから霧状にしたインクを紙に吹きつけるものです。この方式では，インクの色を変えて，カラーのプリントアウトができるという優れたものも開発されていますが，機械の整備や保守にやや手間がかかるようです。

パソコンでもっとも一般的なプリンタといえば，ドットインパクト式です。細いワイヤ状のピンでインクを染みこませたリボンの上から紙を打ち，文字を形づくるものです。Ｘ１のプリンタもこの方式を採用しています。構造上，印字をするときの音がやや大きいのですが，はやい印字速度を得ることができます。ピンは図５－６のように縦に８本並んでいます。８本のピンを横方向に移動させながら，それぞれのピンを出し入れして印字し，文字の形をつくっていきます。

●Ｘ１のプリンタ

Ｘ１用のプリンタCZ-800Pでは，プリンタの内部に専用のメモリを備えています。ですから，Ｘ１の本体からさまざまなコントロールができ，プログラム中でもコントロールができます。

CZ-800Pで使用するのは，幅10インチ（約24cm）のファンフォールド紙というものです。フ

ァンフォールド紙というのは，カメラのフィルムのように両端に紙送り用の穴のあいた紙です。ギアのように爪のついた輪で紙を送る方式をスプロケットフィードと呼び，紙を送る爪はスプロケットピンです。この方法では，感圧紙や裏にカーボンを塗った複写紙を使うこともできますから，複写をとっておく必要のある伝票発行などによく使われます。

この他にもタイプライタと同じように紙をゴムローラではさんで送るフリクションフィードという方法も可能です。この場合には，幅210mm(A４サイズ)の普通紙を使うこともできます。プリンタへの紙の供給は，本体の後ろからでも，本体の下側(底)からでもできるようになっています。

●プリンタを働かせる

次に，実際にプリンタを動かすBASICのコマンド，ステートメントを３つに分けて説明しましょう。

はじめはプログラムリストを打ち出す方法です。長いプログラムを作っていく途中，プログラム全体の流れを確かめたいときや，GOTO文やGOSUB文による分岐があってプログラムの流れが複雑になってきたときなど，プリンタによって印字されたリストがあると便利です。プリンタからリストを打ち出すには，LLISTという命令を使います。LISTとまったく同じようにオペランドを指定します。

LLIST　　全体のリストを出力
LLIST　100，200　　100行から200行までのリストを出力
LLIST，100　　100行までのリストを出力
LLIST　200，　　200行から最後までのリストを出力
LLIST　100　　100行のみを出力

リストの出力と関連して，LFILESがあります。LFILESはFILESの出力をプリンタに行なうものです。１本のカセットテープに何本ものプログラムやデータが記録されているときに，FILESを実行すると，テープに収められたファイル名をすべて画面に表示し，プリンタに出力します。これによってテープの内容の一覧表を作ることができます。

LFILES“ddd：”

として使い，dddはファイルディスクリプタで，CAS：，MEM：，EMM０；～EMM　９こが使用できます（FILESも同じ)。

LLISTによって打ち出したリストで必要なものは，市販のバインダなどを利用して保存しておくとよいでしょう。

２つ目は，実用上もっとも重要なもので，計算結果やデータを打ち出す命令です。これには，LPRINTを使います。使い方はPRINT文とほぼ同じで，プログラム中のTABやUSINGも同じように働きます。PRINT文には各種のコントロールコードがありました。PRINT文とコントロールコードを組み合わせて，さまざまな動作をX１に行なわせることができます。LPRINT文でもプリンタの動作状態を決めるコントロールコードが用意されています。このうち実用的なものをいくつかあげてみましょう。これらのコントロールコードは，

LPRINT CHR$($C_1$, C_2…)

という形で指定します。C_1，C_2の値を変えるとどのような命令となるかを見てみましょう。

1．文字サイズの指定

27，82：普通サイズの文字を指定します。電源を入れた直後はこの状態になっていますから，特に指定する必要はありません。1行の印字数は80文字になります。

27，85：横拡大文字を指定します。表のタイトルなどに使うことが多いようです。1行の印字数は40文字になります。

27，69：縮小文字を指定します。印字数は1行96文字になります。この状態で横拡大指定を行なうと，1行の印字数は48文字になります。

2．改行間隔の指定

作表のところで少しふれたものですが，印字後改行する量を指定するものです。

27，54：電源を入れた直後の状態で，6分の1インチの改行になり，グラフィックキャラクタの間隔が開きます。

27，56：8分の1インチの改行で，グラフィックキャラクタはほぼつながります。

プリンタを動かす3つ目の命令は，画面のコピーを打ち出させるための命令です。グラフィックで描かれた画面をそのままコピーに取るような場合に使う命令です。プログラムの進行中の状態を資料として残したいときや，ゲームの1シーンを残しておいて説明に使うときなどに便利です。

HCOPY n

として使い，nを省略するとテキスト画面（普通の文字が書かれる画面）をコピーします。この場合,ダイレクトモードで行なうとHCOPY命令自身もコピーによって表示されてしまいますから，臨時にプログラム中にHCOPY命令を書いてやるとよいでしょう。

X1の画面構成はこのテキスト画面の他に，3枚のグラフィック画面を加えた4つの画面からできています。nの値によって，次のような組み合わせでコピーを取ることができます。

n＝0：グラフィック1，2，3を合成したもの

n＝1：グラフィック画面1（通常青で描かれているグラフィック）をコピー

n＝2：グラフィック画面2（通常赤で描かれているグラフィック）をコピー

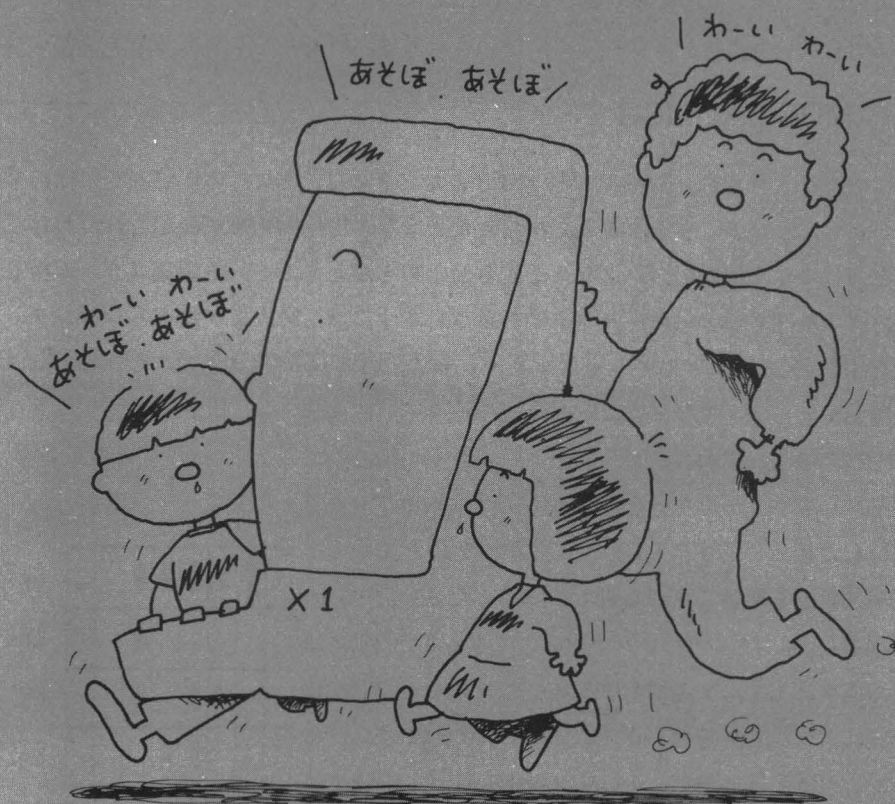
n＝3：グラフィック画面3（通常緑で描かれているグラフィック）をコピー

n＝4：グラフィック画面1から3とテキスト画面を合成したものをコピー

n＝1～3で，通常，と書いたのは，PALET文やCANVAS文で色指定が動かされていないときのディスプレイ上に見られる色のことで，3原色以外の色の場合は，たとえばマゼンタなら青と赤(グラフィック画面1と2)の両方に描かれているわけですから，HCOPY 1でもHCOPY 2でもマゼンタの部分は出力されることになります。

HCOPYを行なっても，CSIZEで指定した拡大文字をコピーすることはできません。

6

X1と遊ぼう

6-1 謎の文字を探る

●PCGってなんだ

CTRLキーとテンキーの／を押してから何かのキーを押してみます。何やら分からないけれど意味ありげな白い模様のようなものが出てきました。X１がこわれてしまったのかと不安になってしまった人は，もう一度CTRLキーと／キーとを押してください。もとの状態に戻ります。

一般に，パソコンでは，表示する文字や記号をキャラクタ・ジェネレータと呼ばれるROM(データの読み出し専用のメモリー)に書き込んであります。X１では，このROMキャラクタジェネレータ（ROMCG）の他に，自由に内容を書きかえることのできるRAMのキャラクタ・ジェネレータ（RAMCG）を持っています。

これを別の名で，**プログラマブル・キャラクタジェネレータ**（PCG）と呼びます。X１の特徴的機能の１つであるPCG機能というのは，キャラクタを自由に設定できる機能のことで，X１用として市販されているゲームのどれをみても，このPCG機能を用いていないものはないといっても良いほどです。いってみれば，このPCGを使いこなさないうちは，本当にX１の機能をフルに活用しているとはいえないでしょう。ここでは，PCGについて見ていきましょう。

●RAMCGとROMCG

X１に電源が入れられ，BASICが動きはじめたときには，まだこのRAMCGには何も書き込まれていません。いま，キーを押して出てきたような，白い四角形が表示されたわけです。もし電源を入れてから何かのプログラムを走らせたあとだとすると，そのプログラム中で定義されていた文字や記号などのキャラクタが表示されます。試しに，アプリケーションテープの中の，TRAIN　X１を動かしてみましょう。

巻き戻したテープをセットし，

```
APSS 2 : RUN "CAS:
```

とキーインしてリターンキーを押します。ひととおりプログラムを動かした後止めます。CTRL＋／キーを押してから，ABBCBBCBBBDと順番にキーを押してみてください。すると，このTRAIN　X１というプログラムの中で走っていた新幹線が画面に描かれます。このプログラムでは，AからDに対応するRAMCGに，新幹線の形を定義していたわけです。

CTRL＋／キーを押すと，ROMCGとRAMCGの切りかえが行なわれます。プログラム中では切りかえを行なうには，CGENという命令を使います。CGEN 1とするとRAMCGとなり，単にCGENとするかCGEN　0　と指定するとROMCGになります。試しにCGEN 0と入力して，リターンキーを押してみてください。文字のかわりに新幹線のパーツが表示されます

〔図6-1〕

16進	10進
&H7C	124
&H4	4
&H14	20
&H18	24
&H10	16
&H10	16
&H20	32
&H0	0

から，注意して入力しなければなりません。これで普通の文字や記号が表示されるようになりました。

RAMCG，ROMCGとも，1つの文字は，横8ドット×縦8ドットで構成されています。これを横1列ずつ8段に分けて考えると，キャラクタ・ジェネレータにセットされるデータの全体も8つのデータに分けることができます。実際にキャラクタ・ジェネレータの中のデータは，横1列ずつ8段分のデータの組み合わせで構成されているのです。たとえば，カタカナの「ア」を例にとると，図6-1のようなデータで構成されています。

枠の外に書かれた数値のうち，左側は「&H」の印でわかるように16進数，右は10進数です。さて，このデータはどのようにして作られているのでしょうか。

まず横1列の8つの枠に左から順に128，64，32，16，8，4，2，1と数をふります。そして，ドットのある枠の位置の数をすべて足します。「ア」の1段目の場合，64+32+16+8+4で124となります。同じようにして「ア」の2段目をみると4となります（図6-2）。

RAMCGにデータをセットし，新しい文字を定義するには，図6-1のような8×8の方眼を用意して，ドットで文字を作るところからはじめます。文字のパターンができたら，先ほどの方法で，8段分のデータを作り出します。次に各段ごとの数値をCHR$関数によって文字列に変換します。

〔図6-2〕

128	64	32	16	8	4	2	1
	●	●	●	●	●		

RAMCGは青，赤，緑の3色に分かれていますから，それぞれの色に対してデータをセットすることになります。

●ひらがなの「あ」を作ろう

具体的な例をあげましょう。ひらがなの「あ」を作ってみることにします（図6-3）。

変数A$に入れておくとすると，A$=CHR$（32，252，40，124，170，146，100，0）となります。

CHR$関数は，与えられた数値をアスキーコードとみなして，それに対応する文字を与える働きをします。

さて，このデータをRAMCGにセットしなければなりません。そのための命令は，

DEFCHR$（n）

です。nは重ねて定義しようとする文字のアスキーコードです。ここでは，ROMCGのカタカナの「ア」に対応するRAMCGに，いま作った，「あ」を定義することにしましょう。「ア」のアスキーコードは，177ですから，

DEFCHR$（177）=A$+A$+A$

とします。A$が3つあるのは，それぞれ，青，赤，緑に同じデータをセットするためです。ここでは，普通の文字や記号と同じように白色で「ア」と表示させたいので，青，赤，緑とも同じデータになるわけで

〔図6-3〕

		■						&h20	32
■	■	■	■	■	■			&hFC	252
		■		■				&H28	40
	■	■	■	■	■			&h7C	124
■		■		■		■		&HAA	170
■			■			■		&H92	146
	■	■			■			&H64	100
								&H0	0

す。

CTRL ＋ ⊘ でRAMCGに切りかえ，カナモードにして ア を押してください。ひらがなの「あ」が表示されます。同じようにしてほかのひらがなも作ってみましょう。

キャラクタ・ジェネレータにセットされているデータの内容は，CGPAT$という関数を使って調べることができます。〔プログラム6-1〕は，アスキーコード177のデータを調べるためのものです。

〔プログラム6-1〕

```
10 CLS
20 A$=CGPAT$(177)
30 FOR I=1 TO LEN(A$)
40 PRINT ASC(MID$(A$,I,1));
50 NEXT:PRINT:PRINT
60 A$=CGPAT$(177)
70 FOR I=1 TO LEN(A$)
80 PRINT HEX$(ASC(MID$(A$,I,1)));
90 NEXT
```

このプログラムを実行してみると画面6-1のように，まず32個の数値が表示されます。最初の8個の数値は，ROMCGに定義されている文字のデータです。カタカナの「ア」のデータと比べてみてください。

〔画面6-1〕

```
ROM CG                              RAM CG青
124 4 20 24 16 16 32 0              32 252 40 124 170 146 100 0
RAM CG赤                            RAM CG緑
32 252 40 124 170 146 100 0         32 252 40 124 170 146 100 0
ROM CG            RAM CG青          RAM CG赤          RAM CG緑
7C414181010200    20FC287CAA92640   20FC287CAA92640   20FC287CAA92640
```

次の8個の数値はRAMCGに定義されている青色のデータ，さらに8個ずつのグループのそれぞれが赤，緑のデータとなっています。これは，ひらがなの「あ」のデータで，文字を白で表示するために，青・赤・緑とも同じデータを用意しましたから，ここでは同じ数値の並びが3回繰り返されています。

その下の文字の並びは，同じデータを16進数で表記したものです。

さて，これまでの作業が理解できれば，RAMCGの各色について，重なったときの色を考えてデータをセットすることによってゲームで使うキャラクタも作ることができます。

図6-4は，TRAIN　X1のB（新幹線の中間の部分）のドットのパターンを3色それぞれについて調べたものです。どこがどんな色になるか考えてみてください。

PCGで作られたキャラクタについても，CSIZEやCFLASH，CREVなどの命令を使うことができます。先ほどの「あ」のように，白で設定したキャラクタに関しては，COLOR命令による色指定も，ROMCGのキャラクタとまったく同じに行なうことができます。

〔図６－４〕

青のデータ		赤のデータ		緑のデータ	
&HFF	255	&HFF	255	&HFF	255
&HFF	255	&H0	0	&H0	0
&H81	129	&H0	0	&H7F	127
&HFF	255	&H0	0	&H0	0
&HFF	255	&HFF	255	&HFF	255
&HEF	239	&HFF	255	&HEF	239
&HD7	215	&HFF	255	&HD7	215
&H0	0	&H10	16	&H0	0

ゲームのキャラクタには，PCGで作った文字を何文字か組み合わせて，大きなキャラクタとして使っているものがあります。ゲーム作りの参考にしてください。

らんだむめも

Errorのチェック

ON ERROR文によってエラー処理ルーチンを作った場合，正しくエラー処理が行なわれるかどうかチェックしておくことが必要です。故意に行なって簡単に発生させられるエラーならよいのですが，多くの場合，エラーを発生させるのにひどく手間がかかります。ところが，ERROR文を使って，希望のエラーをごく簡単に発生させることもできるのです。

ERROR　nとして，nに発生させたいエラーコードを入れます。このステートメントが実行されると，変数の内容などには一切関係なくそのエラーが発生しますから，エラーの処理ルーチンの働きをチェックすることができます。たとえば，

```
INPUT "A=", A : ERROR 11
```

とすると，Aの値に関係なく，ERROR 11が発生したことになります。

6-2　X1はライバル

●パソコンゲームの3つのタイプ

パソコンのさまざまな用途のうちで，忘れてはならないのがみなさんにおなじみのゲームでしょう。パソコン用にもたくさんのゲームが市販されていますが，これらを大きく2つのタイプに分けることができます。まず1つは，「スペース・インベーダー」を代表とする，反射神経を競うタイプのゲームです。これを一般に「リアル・タイム・ゲーム」とか，「反射神経型ゲーム」とか呼びます。X1の優れたグラフィック機能とサウンド機能を活かせば，ゲームセンターのテレビゲームに負けないゲームを作ることができます。もう1つは，「オセロ」のように，思考力に頼るタイプのゲームです。その中でも特に，コンピュータが自ら思考したり判断したりして，人間を相手に対戦するようなものを，「インテリジェント・プレーヤ・タイプ」のゲームなどと呼びます。

また，近頃は両方の要素を取り入れた，「シミュレーション・ゲーム」と呼ばれるものもあります。現実に起こりうる出来事をコンピュータで再現し，これに対する人間の判断力，思考力などを試すもので，これはゲームだけではなく，実際，りっぱに実用として役立っているものもあります。たとえば，パイロット訓練生が，「フライト・シミュレータ」と呼ばれる機械で離着陸の訓練をすることは，みなさんもご存じでしょう。

ここでは，思考型ゲームと反射型ゲームの簡単なものを作ってみましょう。実際にプログラムをキーインしながら，ゲームを作るときに必要な考え方や合理的な処理の方法を学んでください。

●X1に思考力を与える

3×3の枠を書いて，二人で交互に○と×を書き入れていき，どちらかが一直線にならんだら勝ちというゲームがあります。このゲームは，その性格上，先攻の方が勝つことが多いのですが，ゲームを進めるための判断も少なく，また考え方も比較的簡単なので，まずこれをとりあげてプログラム化し，X1を相手に遊べるようにしてみます。

思考型ゲームをプログラム化するときには，そのゲームのルールはもちろん，勝つための条件や，有利なゲーム展開の方法をよく知っておかなければなりません。このゲームのルールは，必ず2人（ここでは人間とX1）が交互に○と×を書いていくということだけです。そして，「勝ち」となるのは，○か×のどちらかが一直線にならんだときです。勝つためには，一直線に並べやすいように自分のマークを書いていくとともに，相手のマークが2つならんだときに

は必ず，３つ目のマスを自分のマークでおさえておかなければなりません。

〔3並べゲームのプログラム〕

```
10 '  ┌─┬─┬─┬──────────────────────┐
20 '  |X|  |X|        3 ﾅﾗﾍﾞ  GAME
30 '  ├─┼─┼─┤
40 '  |X|O|  |        by HAL.Sugawa
50 '  ├─┼─┼─┤
60 '  |O|  |O|   ♣♣ STRATFORD C.C.C ♣♣
70 '  └─┴─┴─┴──────────────────────┘
80 '                  1983 4 2
90 '
100 '
110 WIDTH40:SCREEN0,0:CLS4
120 DIM BAN(3,3),MAN(3,3),COMP(3,3)
130 GOSUB"10円":GOSUB"ﾜｸ"
140 CONSOLE0,25,24,15
150 GOSUB"ｼﾞｭﾝﾊﾞﾝ"
160 GOSUB"CHEC":GOSUB"ﾊﾝﾃｲ":GOSUB"CLR"
170 IF SG=1 GOSUB"MAN":GOSUB"CHEC":GOSUB"ﾊﾝﾃｲ":GOSUB"COMP":GOSUB
"CLR" ELSE GOSUB"CHEC":GOSUB"COMP":GOSUB"CLR":GOSUB"CHEC":GOSUB"
ﾊﾝﾃｲ":GOSUB"CLR":GOSUB"MAN"
180 GOTO 160
190 END
200 LABEL"ﾜｸ"
210 COLOR7
220 FOR I=0 TO 3
230 LINE(0,I*56)-(168,I*56)
240 LINE(I*56,0)-(I*56,168)
250 NEXT
260 FORI=1 TO 3
270 LOCATE I*7-5,22:PRINTI
280 LOCATE21,I*7-4:PRINT I
290 NEXT:RETURN
300 LABEL"MAN"
310 COLOR5:LOCATE24,0:PRINT"ﾄﾞｺﾆ ｼﾏｽｶ ?"
320 LOCATE24,2:INPUT"ﾖｺ",XM:IF XM<1 OR XM>3 THEN BEEP:BEEP:GOTO3
00
330 LOCATE24,4:INPUT"ﾀﾃ",YM:IF YM<1 OR YM>3 THEN BEEP:BEEP:GOTO3
30
340 IF MAN(XM,YM)=1 OR COM(XM,YM)=1 THENBEEP:BEEP:GOTO300
350 CIRCLE(XM*56-28,YM*56-28),20,4
360 MAN(XM,YM)=1
370 K=K+1:RETURN
380 LABEL"CHEC"
390 FOR I=1 TO 3:FOR J=1 TO 3
400 HT(I)=HT(I)+MAN(I,J):HY(I)=HY(I)+MAN(J,I)
410 CT(I)=CT(I)+COM(I,J):CY(I)=CY(I)+COM(J,I)
420 BAN(I,J)=BAN(I,J)+MAN(I,J)+COM(I,J)
430 NEXT
440 HN(1)=HN(1)+MAN(I,I):HN(2)=HN(2)+MAN(4-I,I)
450 CN(1)=CN(1)+COM(I,I):CN(2)=CN(2)+COM(4-I,I)
460 NEXT
470 RETURN
480 LABEL"CLR"
490 FOR I=1 TO 3
500 HT(I)=0:HY(I)=0:CT(I)=0:CY(I)=0:CN(I)=0:HN(I)=0
510 NEXT
520 RETURN
530 LABEL"COMP"
540 IF RND(1)>.8 AND BAN(2,2)=0 THEN XC=2:YC=2:GOSUB680:RETURN
550 FOR I=1 TO 3:IF CT(I)=2 THEN FOR J=1 TO 3:IF BAN(I,J)=0 THEN
 XC=I:YC=J:GOSUB680 :GOTO670 ELSE NEXT ELSE NEXT
560 FOR I=1 TO 3:IF CY(I)=2 THEN FOR J=1 TO 3:IF BAN(J,I)=0 THEN
```

```
 XC=J:YC=I:GOSUB680:GOTO670 ELSE NEXT ELSE NEXT
570 IF CN(1)=2 THEN FOR J=1 TO 3:IF BAN(J,J)=0 THEN XC=J:YC=J:GO
SUB680:GOTO670 ELSE NEXT
580 IF CN(2)=2 THEN FOR J=1 TO 3:IF BAN(4-J,J)=0 THEN XC=4-J:YC=
J:GOSUB680:GOTO670 ELSE NEXT
590 FOR I=1 TO 3:IF HT(I)=2 THEN FOR J=1 TO 3:IF BAN(I,J)=0 THEN
XC=I:YC=J:GOSUB680 :GOTO670 ELSE NEXT ELSE NEXT
600 FOR I=1 TO 3:IF HY(I)=2 THEN FOR J=1 TO 3:IF BAN(J,I)=0 THEN
XC=J:YC=I:GOSUB680:GOTO670 ELSE NEXT ELSE NEXT
610 IF HN(1)=2 THEN FOR J=1 TO 3:IF BAN(J,J)=0 THEN XC=J:YC=J:GO
SUB680:GOTO670 ELSE NEXT
620 IF HN(2)=2 THEN FOR J=1 TO 3:IF BAN(4-J,J)=0 THEN XC=4-J:YC=
J:GOSUB680:GOTO670 ELSE NEXT
630 FOR I=1 TO 3 STEP 2:FOR J=1 TO 3 STEP 2
640 IF BAN(I,J)=0 THEN XC=I:YC=J:GOSUB680:GOTO670   ELSE NEXT:NE
XT
650 FOR I=1 TO 3:FOR J=1 TO 3
660 IF BAN(I,J)=0 THEN XC=I:YC=J:GOSUB680:GOTO670 ELSE NEXT:NEXT
670 K=K+1:RETURN
680 LINE(XC*56-50,YC*56-50)-(XC*56-8,YC*56-8),PSET,2
690 LINE(XC*56-8,YC*56-48)-(XC*56-48,YC*56-8),PSET,2:COM(XC,YC)=
1:RETURN
700 LABEL"ﾊﾝﾃｲ"
710 IF CN(1)=3 OR CN(2)=3 THEN WIN$="ｺﾝﾋﾟｭｰﾀ":GOTO 790
720 IF HN(1)=3 OR HN(2)=3 THEN WIN$="ｱﾅﾀ":GOTO 790
730 FOR I=1 TO 3
740 IF CT(I)=3 OR CY(I)=3 THEN WIN$="ｺﾝﾋﾟｭｰﾀ":GOTO 790
750 IF HT(I)=3 OR HY(I)=3 THEN WIN$="ｱﾅﾀ":GOTO 790
760 NEXT
770 IF K>7 THEN LOCATE24,22:PRINT"ﾋｷﾜｹ!":END
780 RETURN
790 LOCATE24,22:PRINTWIN$;"ﾉ ｶﾁ !!":END
800 LABEL"ｼﾞｭﾝﾊﾞﾝ"
810 LOCATE24,0:CGEN1:COLOR7:PRINTOM$:LOCATE27,1:CGEN:COLOR5:PRIN
T"ｵﾓﾃ = 1"
820 LOCATE24,3:CGEN1:COLOR7:PRINTUR$:LOCATE27,4:CGEN:COLOR5:PRIN
T"ｳﾗ  = 2"
830 LOCATE24,6:INPUT"ﾄﾞﾁﾗﾆｼﾏｽｶ ";UM
840 IF UM<1 OR UM>2 GOTO830
850 PLAY "O5A1+C"
860 FOR I=1 TO RND(1)*10+10
870 UR=I MOD 2:IF UR=1 THEN COIN$=OM$ ELSE COIN$=UR$:UR=2
880 COLOR6:CGEN:LOCATE28,15:PRINT" ! "
890 LOCATE28,16:PRINT" ｉ ":PAUSE1
900 CGEN1:LOCATE28,15:PRINT COIN$:PAUSE1
910 NEXT:CGEN:PLAY"O4+C1A"
920 COLOR6:IF UM=UR THEN SG=1:LOCATE24,20:PRINT"ｱﾅﾀｶﾞ ｻｷ" ELSE S
G=2:LOCATE24,20:PRINT "ｺﾝﾋﾟｭｰﾀｶﾞ ｻｷ"
930 PAUSE10:CLS
940 RETURN
950 LABEL"10円"
960 DEFCHR$(40)=HEXCHR$("0000000000000000071F3F7F7FFFFFFF071F3F7
362F2F2F2")
970 DEFCHR$(41)=HEXCHR$("0000000000000000FFFFFF7F7F3F1F07F2F2F27
2613F1F07")
980 DEFCHR$(42)=HEXCHR$("0000000000000000071F3F7E6CC4EEC0071F3F7
F7FFFFFFF")
990 DEFCHR$(43)=HEXCHR$("0000000000000000FFFFFF7F7F3F1F07FFF7F74
177371F07")
1000 DEFCHR$(44)=HEXCHR$("0000000000000000E0F8FCFEFEFFFFFFE0F8FC
0E66676767")
1010 DEFCHR$(45)=HEXCHR$("0000000000000000FFFFFFFEFEFCF8E0676767
660EFCF8E0")
1020 DEFCHR$(46)=HEXCHR$("0000000000000000E0F8FC7E3663F703E0F8FC
FEFEFFFFFF")
```

```
1030 DEFCHR$(47)=HEXCHR$("0000000000000000FFFFFFFFEFEFCF8E0FF0757
067674F8E0")
1040 OM$=CHR$(40)+CHR$(&H1F,&H1D)+CHR$(41)+CHR$(&H1E)+CHR$(44)+C
HR$(&H1F,&H1D)+CHR$(45)
1050 UR$=CHR$(42)+CHR$(&H1F,&H1D)+CHR$(43)+CHR$(&H1E)+CHR$(46)+C
HR$(&H1F,&H1D)+CHR$(47)
1060 RETURN
```

まず，ゲームをするための枠を描いておきましょう。そして，3×3の枠に縦横ともそれぞれ1から3の番号をつけておき，どのマス目にマークを書くかをこの番号によって指定するようにします。右上のマスなら（1，3）という具合です。この部分をひとつのサブルーチンにまとめて「ワク」というラベルをつけます。

コンピュータに判断をさせるため，ゲーム盤の状態を記憶させなければなりません。そのために3つの配列変数BAN, MAN, COMを用意しておきます。この3×3の配列は，それぞれそっくりゲーム盤と対応しています。変数MANは，人間がマークを書いたところだけを記憶します。同じように変数COMは，コンピュータがマークを入れたところを，そして，BANは，この両方を重ねたもの，つまり，実際の盤面の状態を入れておきます（図6－5）。

〔図6－5〕

人間＝○
コンピュータ＝×

盤面

×		×
	○	
○		

配列MAN

	1	2	3
1	0	0	0
2	0	1	0
3	1	0	0

配列COMP

	1	2	3
1	1	0	1
2	0	0	0
3	0	0	0

配列BAN

	1	2	3
1	1	0	1
2	0	1	0
3	1	0	0

●二次元配列で合理的に

さて，ここでプログラムを見てください。

120　DIM　BAN（3，3），MAN（3，3），COM（3，3）

DIM文で3つの配列変数の配列宣言を行なっていますが，これまでとは少し形が違っています。変数名の後ろのカッコの中には2つの添字が書かれています。このような配列を二次元配列といいます。

DIM　A（3，3）

とすると，変数名Aに3×3個の配列要素が準備されます。それならDIM　MAN（9）とするのと同じではないか，と思う人もいるでしょうが，これまでの添字が1つだけの配列（一次元配列）を使うと，それぞれのマス目に順番に番号を振りあてていかなければなりません。

A	1	2	3	4	5	6	7	8	9	A(9)

たとえば，表6－1を考えてみてください。

この表のマス目に一次元配列を使うと，「B支店のトラック売り上げ台数」をさがし出すにも，

配列の添字を順番にさぐっていくことになります。しかし二次元配列にすると，より合理的にさぐっていくことができます。添字の前の数字を「行」，後ろの数字を「列」としましょう。A支店の総売り上げ台数は，（2，4）でさがすことができます（表6-2）。

表6-1

	本　店	A支店	B支店	合　計
軽自動車	12	3	5	20
小型自動車	15	6	5	26
トラック	2	0	2	4
合　計	29	9	12	50

表6-2

	本　店	A支店	B支店	合　計
軽自動車	(1，1)	(1，2)	(1，3)	(1，4)
小型自動車	(2，1)	(2，2)	(2，3)	(2，4)
トラック	(3，1)	(3，2)	(3，3)	(3，4)
合　計	(4，1)	(4，2)	(4，3)	(4，4)

このゲームでも，それぞれのマス目に1から9までの番号をふるよりも，二次元配列を使ったほうが合理的なプログラムを組むことができます。

●箱の中に記憶を入れよう

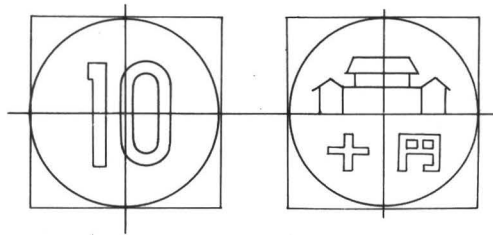
これで盤面の状態を記憶させるための箱が用意できましたから，次は箱の中へ記憶を入れる方法です。先ほどの表のように，マークの書かれたところに対応する配列要素を1に，何も書かれていないところを0として区別します。これで盤面上の状態が数値によって表現されることになります。

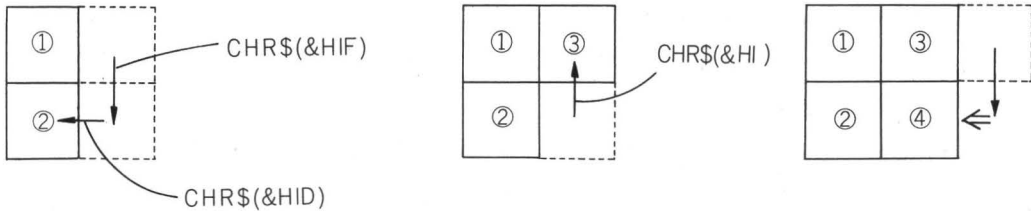
人間が，キーボードからマークを入れるマス目の縦と横の番号を入力する部分を，サブルーチン“MAN”としてまとめました。320行と330行の後半で，誤った番号が入力されないようにしています。このような処理をしておかないと，ありえないデータに対しても，そのまま受けつけられてしまい，盤の外にマークがついてしまうといった不都合が生じます。さらに，大切な判断を340行で行なっています。変数MAN（XM，YM）と変数COM（XM，YM）の中身のチェックです。縦と横の番号を入れようとするとき，2つのうちどちらかの配列変数の中身が1になっていたら，そこにはすでにマークが書かれているわけですから，もう一度入れ直すようにしてあります。BEEP命令でブザーを鳴らし，もう一度入れるように注意をうながすこともしています。このゲームでは，すべてMAN, COM, BANの3つの配列変数の中身を調べて，盤面上の状態を判断するようになります。

350行では，実際に盤面上に○を書いています。縦横の番号から，CIRCLE文の中心を割りだしています。

●先攻，後攻を決める

次に，先攻・後攻を決めるためのサブルーチン「ジュンバン」を見てください。ここでは，10円のコインを投げて（?）その裏表によって決めるようにしました。コインの裏と表の図案を，950行からのサブルーチン「10円」でPCGを使って作っています。このコインは，図6-6のように4文字分のキャラクタで作られています。PRINT文で画面に描くときに，1回のPRINT文で全体を表示できるように，つまり4文字分のキャラクタを1つにまとめてしまえるように工夫してあります。

〔図6-6〕

1040行と1050行で使っているカーソル移動コードがそれです（図6-7）。１文字分のデータの次にカーソルを移動させるためのコードを入れてつないでいます。こうしておけば，その文字変数の内容がPRINTで表示されるときには，カーソルの移動が行なわれて，１つの文字変数によって２行にわたるキャラクタが表示できるというわけです。

〔図6-7〕

順番を決めるには，FOR～NEXTの終了値を乱数にしておき，制御変数が奇数のときには表を，偶数のときには裏を表示するようにしておきます。このFOR～NEXTのループで10円玉が回転する回数が決まりますが，最低でも10回はくるくると回るようになっています。FOR～NEXTをぬけたときの制御変数が偶数か奇数か，つまり，コインが表で止まったか裏で止まったかを調べます。裏表どちらにするかは，変数UMに数値で入力するようにしてありますから，制御変数URがUMと一致していれば人間の先攻に，異なっていればコンピュータの先攻となります。先攻か後攻かというデータは，変数SGに代入しておいてメインルーチンでゲームスタートの順番の判断に使っています。

次は，いよいよコンピュータに考えさせる方法について説明します。

6-3　X1に知能を？

●X1を対戦相手にしたてよう！

このゲームのように，コンピュータに対戦の相手をさせるには，自分がゲームをしているときにどんな考え方をしているかをよく分析することが必要です。それをうまくプログラム化しなければ，X1は優秀な対戦相手にはなってくれません。人間が考えるその考え方をX1に与える，いわばX1に知能を与える作業が，ここで行なうプログラミングの中心です。

このゲームの考え方を書き出してみます。

1．最初に自分のマークを置くときは，四角に置く。

2．自分のマークが2つ並んでいて，残りが空いていれば必ずそこに打つ。

3．もし相手のマークが2つ並んでいて，残りの1カ所が空いていたら必ずそこに打って，相手の動きを止める。

4．2と3の条件にあうところがなければ，他に空いているマス目のうち斜めの角に打つ。

5．4もだめなら，残りの適当なところに打つ。

大まかにまとめると以上のようになります（4，5については，まだ考えなくてはいけない部分もありそうですが，一応ここまでの考え方で進めていくことにします）。

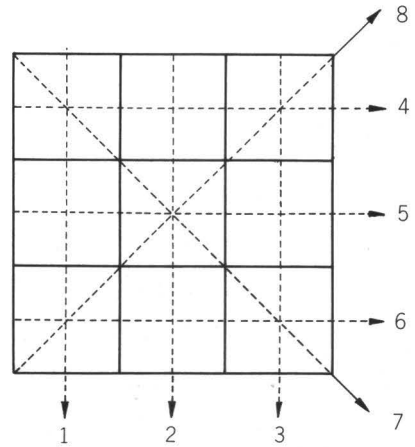
コンピュータに判断をさせる順序も，ここに書き出した順でうまくいくでしょう。1の判断は一応後に回すことにして，2の判断について考えてみます。これからあとは，コンピュータの側に立って判断の条件を考えていきましょう。コンピュータに則していきますから，「自分」というのはコンピュータ自身のことになります。

●自分のマークを探す

まず，自分のマークが2つ並んでいるところをさがし出さねばなりません。自分のマークの置かれた位置を記憶している配列変数COMの内容を調べます。この配列の要素のうち，1になっているところが自分のマークがすでに書かれているところですから，図に示す8通りの方向にならんだ配列要素を加えていきます(図6-8)。この作業をしているのが，サブルーチン「CHEC」の，410行と，450行です。図6-8の1から3の縦の並びが配列変数CT（1）からCT（3）に，4から6の横の並びがCY（1）からCY（3）に，7と8がそれぞれCN（1）とCN（2）に代入されます。これらの配列変数については，配列宣言を行なっていませんが，X1では特に宣言されていない配列については，要素の数は10として扱われます。言いかえれば要素の数が10以下のときは配列宣言をしなくてもよいことになります。配列変数CT，CY，CNの中の値は最初は0ですが，配列変数COMの中に代入された値が0でなければ，つまりどこかに自分のマークが置かれていればその数に応じてCT，CY，CNの値も増えることになります。

サブルーチン「COMP」の中では，この8つの配列変数を調べ，もしその値が2になっているものがあれば，ということは自分のマークが2つ並んでいるところがあればですが，その配列

〔図6-8〕

変数に対応する列をもう一度確める作業をします。自分のマークだけではなく，相手のマークの有無も調べなければなりません。両方のマークの有無が同時に記憶されている，配列変数BANを調べます。配列変数BANには，2人のプレーヤのどちらかのマークが置かれていれば1，どちらも置かれていなければ0の値が代入されていますから，0のところがあればそこには自分のマークを書くことができるわけです。×を書くサブルーチンを呼び出してマークを書くと同時に，その位置に対応するCOM（3，3）の要素を1にします。

これでその1列に×が3つならんだわけですから，自分の勝ちとなります。

勝負の判断は，サブルーチン「ハンテイ」で，合計が3になっている列を探すことによって行なわれています。もし，配列変数CT，CY，CNの値が2になっていても，残りの位置が0でなければ，そこは相手（人間）によって止められているわけです。550～580行の少し複雑な条件判断でこの確認をしています。ここで使われる変数XC，YCは×を書く座標位置を決めるために使われます。確認の結果3つのマークを並べる，という条件で打つところがなければ，次の判断に進みます。

●相手のマークを調べる

今度は，相手のマークについて調べなければなりません。2つ並んだところがないかどうかをさがします。これもまったく同じ方法で，サブルーチン「CHEC」を使うことによって，配列変数MANの中身を調べ，それぞれの並びを配列変数HT，HY，HNに代入しておきます。もし配列変数の内容が2になっている列があって残りのところが0なら必ずその位置に自分のマークを打っておかないと，次の回で相手側の勝ちになってしまいます。590から620行でこの判断を行なっています。

これも打つところがなければ，斜めの角で空いているところをさがします。630と640行がこの四角の調査と判断を行なう部分です。

盤の四角も空いていなければ，残りのマス目のうち空いているところに打つことにします。650と660行がこの判断です。

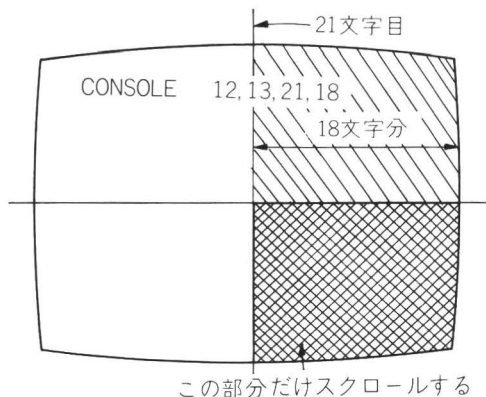
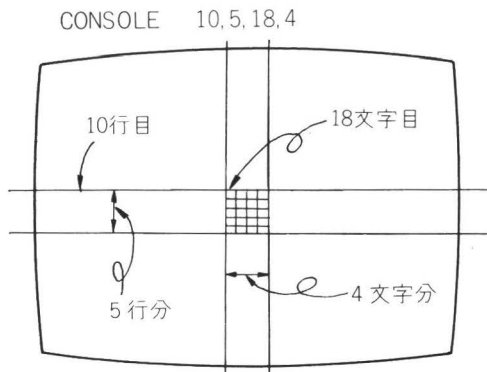
これで，2から5までの判断を行なうことができました。それぞれの場面でいろいろな判断をしていますが，考え方はどれも同じです。配列変数を使うといかに合理的に調査や判断が行なえるかがわかります。1については，はじめのときは2と3はあり得ないわけですから，1番先に判断を行ないます。

●そのほかのプログラミングテクニックは？

全体のプログラムは，これまで見てきたことでわかるように，細かいサブルーチンの集まり

でできています。その中でも，コンピュータに判断させる，「COMP」は，一番大きいものになっています。人間なら一目で判断できることもプログラム化するとかなりの量になることもわかります。人間の頭脳の優秀さを表わしているといえるでしょう。

170行では，先攻，後攻の順番を示す変数SGの値により，サブルーチンを呼び出す順序を変えるという作業をしています。サブルーチン「CLR」は，配列変数CT，CY，CNの内容を毎回すべて0に戻す操作をしています。必要なときに，サブルーチン「CHEC」を呼び出し，それぞれの配列変数の内容が0であれば，盤の状態についての最新の情報を得ることができます。逆にいえば，この設定をしておかないと正しい判断をさせることはできなくなります。勝敗の判断をするサブルーチン「ハンテイ」も，この配列変数に基づいて判断を行なっていますから，必ずこのサブルーチン「CLR」を呼んでから「ハンテイ」を呼ぶようになっています。

このゲームは，一見すると中央を先に取った方が八方へのつながりがあって良いように思えますが，実は先に1カ所角を取るのが必勝法なのです。しかし，人間であってもときにはいきなり中央を取ることもあります。コンピュータにもこのような人間の動作をまねさせるために，このプログラムでは540行で，RND関数を使ってコンピュータに気まぐれの動作をさせています。1：4の割合でいきなり中央を取るようにしてあるのです。RND（1）によって得られる数値は，0以上1未満の数値（0<RND（1）<1）ですから，10回に2回はあたかも気まぐれを起こしたかのようにコンピュータは中央のマス目をねらってきます。

〔図6-9〕　〔図6-10〕

細かいテクニックでは，画面のスクロールエリアを決めるCONSOLE文が使われています。通常，リストをとったりした場合，一番下の行まで行くと画面全体が上にせり上がって次の行が表示されます。これを「スクロール」と呼び，CONSOLE文はこのスクロールの範囲を決める命令であることは前に説明しました。ここでは，CONSOLE命令でこれまでとは少し違った範囲指定を行なっています。X1ではスクロール範囲は，上下方向だけでなく，左右方向にも指定することができるのです。図6-9のように，画面のすみの部分だけにスクロール範囲を指定することもできるのです。また，図6-10のように，画面の任意の位置にちょうど窓をあける感じでスクロール範囲を作ることもできます。CLS文によって，テキスト画面を消去するときにも，このスクロール範囲の中だけで行なわれますから，うまく使うと効率の良いプログラム作りができます。このプログラムは，「コイン投げ」のあと，930行で画面の消去を行なっています。

ところが140行でスクロールエリアを画面の右側の方だけと決めているため，表示したゲーム盤につけられた番号は消えることがありません。また，ここで使っているPAUSE文は，プログラムの進行に待ち時間をつくるための命令で，後に書く数値が待ち時間の長さを決めます。１を指定すると約10分の１秒間停止してから次のステートメントに移ります。FOR～NEXTの空まわりループよりも，正確な待ち時間を指定することができます。その他，このプログラムでは，IF～THEN～ELSEを多用していますから，もう一度IF～THENの条件判断，特にELSEの働きを復習しておいてください。

6-4 X1でゲーム

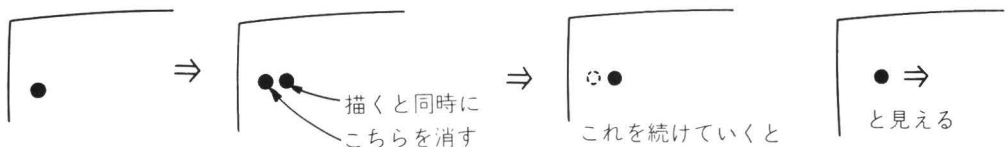
●反射神経型のゲーム

あなたのX1は，知能があるように上手な判断をしていますか？

思考型のゲームに対して，もう1つのタイプのゲームを反射神経型，あるいはリアルタイム・ゲームと呼びます。インベーダーゲームのように敵が攻撃してくるのを迎え撃つタイプのものや，テニスのような球技をゲーム化したもののことです。敵の攻撃やボールの動き，それにともなう状態の変化などに対応する，反射神経の鋭さやすばやい判断力などを競うものです。こういったタイプのゲームは，ゲームセンターでおなじみですし，X1用としてもずいぶんたくさんのものが市販されています。ここでは，このようなタイプのゲーム作りに欠かせないいくつかのテクニックと，基本的な考え方を学んでいきましょう。

反射神経型のゲームでは，インベーダーやボールなどを画面上で動かさなければなりません。これにはどのような方法を使えばよいかをまず最初に考えてみましょう。

はじめの位置に，たとえば●が描かれています。これを右に動いていくように見せるには，今ある位置の右隣りに，●を描きます。それと同時に，前の位置にあった●を消してしまいます。具体的には，LOCATE文のパラメータを変化させながら，PRINT文によって●を描き，Xのパラメータを1減らしたところに，スペースを描いて前の●を消せばよいわけです（図6-11)。

〔図6-11〕

〔プログラム6-2〕

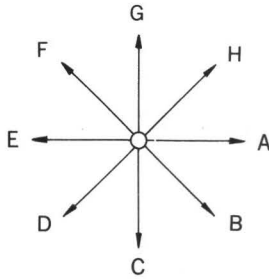
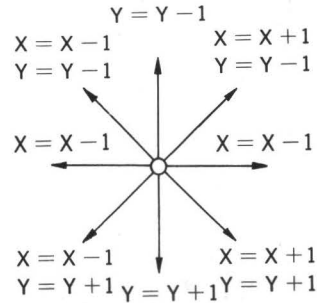
```
10 WIDTH40:CLS
20 FOR X=1 TO 39
30 LOCATEX,12:PRINT"●":LOCATEX-1,12:PRINT" "
40 NEXT
```

●ボールの描き方

この〔プログラム6-2〕を実行してみると，ほとんど目に止まらぬくらいのはやさで●が動いていきます。このプログラムでは画面の左から右へまっすぐに動いていくだけですが，実際のゲームでは斜めに動かしたり，下から上へ，右から左へともっと自由な動きが要求されます。これらの動きを，図6-12のように8方向に分けて考えてみます。まず，Aの方向については見たとおりです。B方向には，X，Y2つの値を同時に増やしてやると動かすことができます。D方向はXを減らしながらYを増やすことになります。例1のプログラムを書き直してこれらの方向に●を動かしてみてください。

図6-12を書き直してみると，図6-13のように，8方向への移動をLOCATE文のパラメータの

変化で作り出せることがわかります。さらに正反対の方向への移動は，式の右辺にある＋と－とを逆にすればよいということもわかります。

〔図 6 -12〕

〔図 6 -13〕

今ではもう古典的と感じられるテレビゲーム「ブロックくずし」で，みなさんも遊んだことがあるでしょう。ブロックくずしのボールの動き方を思い出してみてください。

ボールが壁に当たってはねかえるときの変化は図のようになっています。上下の壁に当たったときは，パラメータのYの値の変化のしかたをそれまでと逆にしてやると良いですし，左右の壁に当たったらXの値の変化のしかたを逆にしてやれば良いわけです。

さて，これだけのことがわかったのですから，ここでは「ブロックくずし」ゲームを作ってみることにしましょう。もはや，古典とはいえ，古典的なゲームも味わいぶかいものです。

●ボールは壁に当たったか？

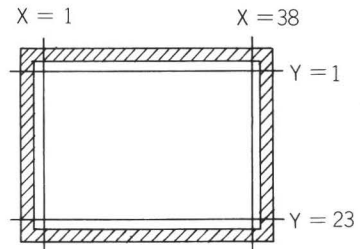
次に，ボールが壁に当たったかどうかの判断をさせるための方法を考えてみることにします。

方法は2つあります。まず1つは，CHARACTER$という関数を使って，壁があるかどうかを調べる方法です。CHARACTER$は，指定した座標に表示されている文字を値として取り込む関数です。しかし，CHARACTER$を使う方法では，球のまわりの8カ所はすべての座標を調べなければなりません。面倒であるだけでなく，このゲームのようにある程度はやい動きを要求する場合には，8カ所を次々と調べることでかなり速度が落ちます。ですからここではこの方法は不向きです。もう1つの方法は，球のある位置の座標の数値から判断する方法です。40文字モードで画面いっぱいに壁を描いた場合，球の動くことができる座標の範囲は，1≦X≦38，1≦Y≦23になります。これを利用してやれば，短い処理で壁の有無の判断をすることができます。Xが，1または38になったときには，球の動くX方向の向きを変えてやればよいのです。また，Yが1または23になったときには，Y方向の向きを変えてやります。これを利用して組み立てたのが〔プログラム6－3〕です。

20行で周囲の壁を作ります（図 6 -14）。40行と50行は，変化させる方向を決める変数を作る部分です。まず0から2までの数を乱数で作り出し，その数から1を引くことで，－1，0，1の3つの数値を作っています。さらに，こうして得られた値が0のときにはもう一度やり直して，－1か1を作っています。乱数では負の値を作り出すことができないので，このような操作をしています。こうして作られた変数XRとYRは，球の最初の位置を表わすX，Yに加えられて球の進行方向を決めています。

〔プログラム 6-3〕

```
10 WIDTH40:CLS
20 COLOR4:LINE(0,0)-(39,24),"▒",B
30 X=20:Y=12:XC=X:YC=Y
40 XR=INT(RND(1)*3)-1:IF XR=0 GOTO40
50 YR=INT(RND(1)*3)-1:IF YR=0 GOTO50
60 ' ROOP
70 X=X+XR:Y=Y+YR
80 IF X=<1 OR X>=38 THEN XR=-XR
90 IF Y=<1 OR Y>=23 THEN YR=-YR
100 LOCATEXC,YC:PRINT" "
110 XC=X:YC=Y
120 COLOR7:LOCATEX,Y:PRINT"●"
130 GOTO60
```

〔図 6-14〕

画面いっぱいに
壁を描いた時

80行と90行で，壁に当たったかどうかに相当する判断を行ない，条件が成り立っていれば今までマイナスだったものをプラスに，プラスであったものをマイナスにしています。この理屈は前に説明しました。壁に当たった球は，反射して飛び続けます。変数XC，YCには移動し続ける球の1つ前の位置が代入されることになり，新しい球を描く直前に前の位置の球を消しています。このプログラムを動かしてみると，球の動きもそのはやさもゲームとして使えそうです。これを基礎に，「ブロックくずし」を作っていくことにします。

●音も出そう

ゲームらしくするために，球が壁に当たってはねかえるときに音を出してみることにしましょう。40行と50行の後にPLAY文で音を出すように追加すればよいわけです。ところが，このような処理をすると音が出ている間，球の動きが止ってしまい，ゲームとしては不自然になってしまいます。PLAY文は，文字列で与えられたデータを，音を出すために必要な形に変換してからPSG（プログラマブル・サウンド・ジェネレータといって音を出すためのLSI）に送っているため，このようなことが起こるのです。SOUND文を使えば，PSGに直接データを送ることができますから，この時間を省くことができます。また，SOUND文では，一度データを送れば特に操作しないかぎり音は出続けていますから，音を出しながら次の命令に進むことができます。

〔プログラム 6-4〕では，15行であらかじめ音程と音を出すチャンネルを決めておき，必要なときに音量を上げ，その音が聞こえるようにしています。80行と90行のSOUND 8，15がそれで，このステートメントが実行されると音が出はじめ，プログラムはすぐに次の行へと進みます。そして，130行までやって来たときにSOUND 8，0という命令で音量を0にします。

〔プログラム 6-4〕

```
10 WIDTH40:CLS
15 SOUND0,28:SOUND1,1:SOUND7,62
20 COLOR4:LINE(0,0)-(39,24),"▒",B
30 X=20:Y=12:XC=X:YC=Y
40 XR=INT(RND(1)*3)-1:IF XR=0 GOTO40
50 YR=INT(RND(1)*3)-1:IF YR=0 GOTO50
60 ' ROOP
70 X=X+XR:Y=Y+YR
80 IF X=<1 OR X>=38 THEN XR=-XR:SOUND8,15
```

```
90 IF Y=<1 OR Y>=23 THEN YR=-YR:SOUND8,15
100 LOCATEXC,YC:PRINT" "
110 XC=X:YC=Y
120 COLOR7:LOCATEX,Y:PRINT"●"
130 SOUND8,0:GOTO60
```

6-5 ようこそ，ゲームセンター「X1」へ！

●ラケットと球の動きは？

ブロックくずしの基本的な球の動きは，これまでのプログラムの応用でよいようですから，先に画面のレイアウトを決めておきましょう。40文字モードでゲームを行なうことにして，画面を図６-15のように分割しました。

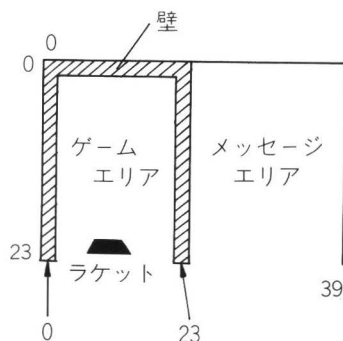
球の動く範囲は〔プログラム６-４〕の数値を一部変えるだけで良いことがわかります。

〔図６-15〕

ただし，今度は一番下の壁は不要となります。球がここまで来たら，ラケットで球を受けとめ，はじき返してやらねばなりません。

壁は，LINE文のBOXで四角い枠を描いた後に，一番下の壁の部分を消すという方法を使いました。

次に，ラケットを左右に動かす方法を考えます。

プレーヤーが何かのキーを押してラケットを動かすのですから，そのキーが押されているかどうかの判断がまず必要になります。動きのあるゲームですから，INPUT文は使えません。また，INKEY$関数は１文字分しか受け付けませんから，押している間ずっとラケットを移動させるわけにはいきません。INKEY$（０）は，連続してキー入力を受けつけますが，ここでは別の方法を使います。

●STICK関数を使って

ゲームにとても便利な関数が１つありますから，これを使うことにしましょう。それはSTICK関数です。これは，テンキーやジョイスティックから入力される値を与える関数です。STICK関数は

STICK(n)

として使います。nは０，１，２の３つで，０のときはキーボードのテンキー，１，２はそれぞれ，X１本体の後ろにあるジョイスティック端子の１と２に対応します。求められる数値は図６-16のようになっていて，キーボードのテンキーの５を中心とした８方向に対応する数値が得られます。もし何も押していない（スティックを倒していない）ときには値は０になります。次のプログラムで試してみましょう。

```
10 S=STICK(0)
20 PRINT S;:GOTO10
```

テンキーのどれかを押している間，その数値が変数Sに代入されます。

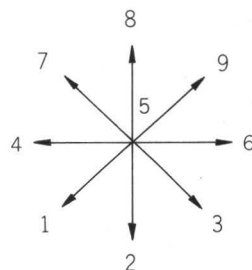
ラケットの移動には，テンキーの[4]と[6]とを使い，[4]のキーを押すと左へ，[6]を押すと右へラケットを移動することにします。実際にラケットを動かすには，PRINT文を使うこともでき

〔図 6 -16〕

7	8	9	
4	5	6	
1	2	3	

ますが，ここでは，ゲームでしばしば使われるGET@，PUT@という命令を用います。この方法を知っておくと，ラケットよりももっと大きなものを動かしたいときや，グラフィックで描いたものを動かすときに応用することができます。

〔プログラム 6 - 5〕はPRINT文を使ってラケットを動かす例です。

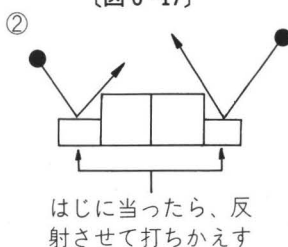
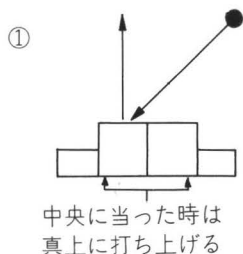
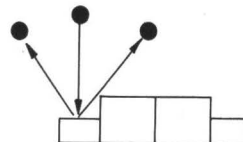
〔プログラム 6 - 6〕では，35行のGET@であらかじめ画面に描いておいたラケットの絵を，配列変数RA%に取り込みます。GET@関数は，後に指定した，X，Y座標の範囲の画面の状態を取り込む働きをします。この範囲を変えることによって何行にもわたる絵を一度に取り込むことができるのです。また，配列の後にパレットコード(色コード)をつけると，グラフィックで描かれたものを取り込むことができます。PUT@はこの逆の働きをする関数です。配列名の後に，PSETなどの指定をつけると，グラフィックによる絵を描くことができます。この2つを組み合わせると，かなり大きな絵を動かすことができます。

「ブロックくずし」の基本となる球の動き，ラケットの動きをプログラムすることができましたから，後は細かい部分を考えていきます。

ラケットに球が当たったときの動きですが，次の図 6 -17のようにしてみます。

このような動きをさせるには，②では，壁に当たったときとまったく同じように球の動く方向を示す変数YRの値の正負を反対にしてやるだけです。①では，YRの正負を反対にして，さらにX方向の向きを示すXRの値を 0 にします。③は，②の特別な場合と考えればよいでしょう。球が当たったときのXRの値が 0 ですから，乱数を使ってXRの値を決め直してやればよいわけです。

〔図 6 -17〕

②

はじに当ったら、反
射させて打ちかえす

③　もし真上から落ちてきたら、方向を変える

〔プログラム 6 - 5〕

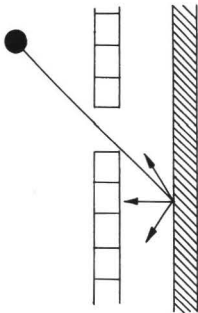
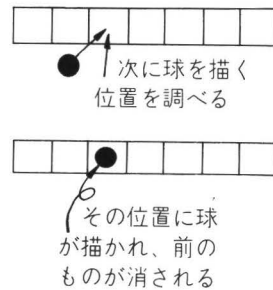
```
10 '
20 X=20:Y=12
30 RA$=" ▬▬ "
40 S=STICK(0)
50 IF S=0 THEN XS=0
60 IF S=4 THEN XS=-1
70 IF S=6 THEN XS=1
80 X=X+XS
90 IF X=<1 THEN X=1
100 IF X>=22 THEN X=22
110 LOCATE X,20:PRINT RA$
120 GOTO 40
```

〔プログラム6-6〕

```
10 DIM RA%(8)
20 X=10 Y=24
30 LOCATE X,Y:PRINT" ▂▂▂ ";
35 GET@(9,24)-(13,24),RA%
40 S=STICK(0)
50 IF S=4 THEN XS=-1
60 IF S=4 THEN XS=-1
70 IF S=6 THEN XS=1
80 X=X+XS
90 IF X=<1 THEN X=1
100 IF X>=22 THEN X=1
110 PUT@(X,Y)-(X+5,Y),RA%
120 GOTO 40
```

●正確にはねかえる球はつまらない

今度は，ブロックに球が当たったときの動きの変化ですが，これは，まったく予測ができないようにしておいた方がおもしろいので，完全に乱数で決めることにします。ただし，変数XR，変数YRの値とも0になってしまうと球が動かなくなりますから，0になったらもう一度やり直すようにします。

球がブロックの列を抜けて，上の壁に当たったときは，球の動きを当然下向きにしなければならないので，球を反転させるためにXRの値だけは乱数を使って決めます（図6-18）。

〔図6-18〕　〔図6-19〕

球が当たったときの判定は，壁以外はCHARACTER$関数を使って調べます。このとき，いつこの判定をするかがポイントになります。ここでは，次に球を描くべき位置が決まったところで，球を描く前にその位置の画面の状態を調べるようにしました（図6-19）。このようにすると，自然な動きでブロックを消すことができます。ラケットの場合は，球によって消されても，すぐにPUT@文で描き直されますから，不都合はありません。

〔プログラム6-7〕

```
10 DIM RA%(8)
20 WIDTH40:SCREEN0,0:CLS4
30 COLOR4:LINE(0,0)-(23,24),"■",B
40 LINE(0,24)-(23,24)," "
50 LOCATE10,24:COLOR6:PRINT"▂▂▂";
60 GET@(9,24)-(13,24),RA%
70 SOUND0,28:SOUND1,1:SOUND7,60
80 XB=10:YB=24:XM=10:YM=24:XC=XB:YC=YB:BALL=8
90 XR=INT(RND*3)-1:IF XR=0 GOTO90
100 YR=-1
110 LOCATE2,3:COLOR3:PRINTSTRING$(20,"■")
120 LOCATE2,4:COLOR5:PRINTSTRING$(20,"■")
130 LOCATE2,5:COLOR6:PRINTSTRING$(20,"■")
140 COLOR7:FOR I=1 TO BALL:LOCATE30+I,6:PRINT"●";:NEXT
```

```
150 GOSUB"ｺﾝﾄﾛｰﾙ"
160 ' ﾗｹｯﾄ
170 S=STICK(N):IF S=0 THEN XS=0
180   IF S=4 THEN XS=-1 ELSE IF S=6 THEN XS=1
190   XM=XM+XS:YM=YM+YS
200   IF XM=<0 THEN XM=0 ELSE IF XM>=19 THEN XM=19
210 PUT@(XM,24)-(XM+5,24),RA%
220 ' BALL
230 XB=XB+XR:YB=YB+YR
240 IF XB=<1 AND XR=-1 XR=1:SOUND8,15
250 IF XB>=22 AND XR=1 XR=-1:SOUND8,15
260 IF YB=<1 THEN YR=-YR:SOUND8,15:IF XR=0 THEN XR=INT(RND*3)-1
270 IF YB=25 THEN GOSUB"ﾊﾝﾃｲ":GOTO160
280 C$=CHARACTER$(XB,YB)
290 IF C$<>" " THEN SOUND8,15
300 LOCATEXC,YC:PRINT" ";
310 XC=XB:YC=YB
320 LOCATE XB,YB:COLOR7:PRINT "●";
330 IF C$="█" GOTO 400
340 IF C$="▄" THEN YR=-YR:XR=0
350 IF C$="▂" THEN YR=-YR:IF XR=0 THEN XR=INT(RND*3)-1
360 IF C$<>"▄" GOTO400
370 IF YB=3 THEN PO=PO+20:P=P+1 ELSE IF YB=4 THEN PO=PO+10 :P=P+
1 ELSE IF YB=5 THEN PO=PO+5:P=P+1
380 IF P=60 THEN FOR B=0 TO 1000:NEXT:BEEP0:END
390 XR=INT(RND*3)-1:YR=INT(RND*3)-1:IF XR=0 OR YR=0 GOTO390
400 LOCATE30,4:PRINT "POINT";PO
410 SOUND8,0:GOTO 160
420 LABEL"ﾊﾝﾃｲ"
430 PLAY"::O2A7"
440 LOCATE30,6:PRINTCHR$(5)
450 BALL=BALL-1:IF BALL=0 THEN END
460 FOR I=1 TO BALL
470 LOCATE I+30,6 :PRINT"●":NEXT
480 XB=XM:YB=YM:XR=INT(RND*3)-1:YR=-1
490 PAUSE10:LINE(0,24)-(23,24)," "
500 RETURN
510 LABEL"ｺﾝﾄﾛｰﾙ"
520 LOCATE2,16:COLOR5:PRINT"ｽﾍﾟｰｽ ﾊﾞｰｶ"
530 LOCATE2,18:PRINT"ﾄﾘｶﾞｰ ﾎﾞﾀﾝ ｦ ｵｼﾃｸﾀﾞｻｲ"
540 S0=STRIG(0):S1=STRIG(1):IF S0=0 AND S1=0 GOTO540
550 IF S0=-1 THEN N=0 ELSE IF S1=-1 THEN N=1
560 LINE(2,16)-(22,18)," ",BF
570 RETURN
```

●さて仕上げは？

以上のことをまとめて，プログラムとしたのが〔プログラム6-7〕です。このゲームは，ジョイスティックとキーボードのどちらでも使えるようにしてあります。どちらを使うかを決めるのが510行からのサブルーチンです。STRIG関数を使っています。

この関数は，スティックのトリガーボタン，またはキーボードのスペースバーの状態を調べるものです。

STRIG(n)

として使います。nの値はSTICK関数と同じで，0ならキーボード，1と2はスティック1と2のトリガーボタンに対応します。これらが押されていると，この関数の値は－1に押されていなければ0になります。ここでは，キーボードとスティック1の両方を調べ，押されている方

を使うように変数Nの値を決めます。変数Nの値はそのままメインルーチンに受けつがれてSTICK関数の引数（カッコの中の数値）として使われます。

このプログラムでは，今までのようなサブルーチン分けをしていません。これは，速度の低下を防ぐためです。サブルーチン分けしたプログラムは，大変わかりやすく，便利な点も多いのですが，毎回サブルーチンに飛んでいって処理をするため，この往復の分だけ余計な時間がかかります。このようなリアルタイムゲームでは，なるべく無駄な処理や分岐を省いて，速度を上げる方法を考えなければなりません。FOR～NEXT文なども，速度を低下させる大きな原因になっていますから，うまく使い分けてください。

このプログラムでは，「’」(アポストロフィー，REMの省略形)を使って，今まで説明してきた要所にコメントをつけてありますから，本文と合わせてその働きを理解してください。

他のゲームを作るときも，ここまで説明してきたような基本的な考え方はほとんど共通しています。参考にしてみなさんも楽しいゲームを作ってみてください。

6-6　ちょっとハイテク，エラー処理ルーチン

●エラーを出しても大丈夫？

私たちがBASICでプログラムを作って，走らせたときによく発生するのが，前にも出てきたSyntax errorや，Illegal function call errorです。

これらのエラーは，命令文のつづりやパラメータの使い方がおかしいために発生するエラーです。プログラム自体に問題があるわけですから，修正を加えていくことになります。こういった修正を加えていって，完成されたプログラムでも，プログラムを使う人が誤ったデータを入力するなどの誤操作によってエラーストップすることがあります。

このような場合でもプログラムの実行をストップさせず，実行を続けていけるようにするのが，エラー処理命令です。次のプログラム例をみてください。

```
10  INPUT "A",  A
20  INPUT "B=",  B
30  C=A/B
40  PRINT  C
50  GOTO  10
```

これは，AとBに2つの数値をキーボードから入力し，その結果を表示するものですが，Bの値に0を入力すると，「Devision by zero」エラー（エラー11）と表示して止まってしまいます。もしこれが大きなプログラムの途中で起きたものだとすると，大変面倒なことになります。このような場合，あらかじめこうした入力ミスをしそうなところをチェックするとともに，起こりそうなエラーを想定して，エラー処理ルーチンを作っておきます。すると，プログラムの実行をストップさせずに続けることができます。

エラー処理ルーチンへ実行を移すには，ON ERROR GOTOという命令を使用します。このステートメントは，プログラムの先頭に置き，とにかくエラーが起きたら，指定した行番号へジャンプするように指定しておくものです。そして，エラー処理ルーチンの中では，エラーの発生した場所とエラーの種類によって，対応する方法を設定しておき，RESUME文によってメインルーチンに戻るようにしておきます。

●ERRとERL

X1では，システム変数（関数の一種です）としてERR，ERLが用意されていて，プログラムの実行中にエラーが発生するとERRにエラーコードが，ERLにエラーの発生した行番号が入ります（エラーコードとエラーの内容については，巻末の表を参照してください）。前出のプログラム例の場合，ERRとERLの値をPRINT文で調べてみると，それぞれ11と30になっていることがわかります。

この場合，Bに０の値が入ったために30行でエラーコード11のエラーが発生したのですから，これを解消（保証）するには，数値を入力し直すことになります。この処理を，10000行からのエラー処理ルーチンにまとめてみることにしましょう。プログラムのはじめのところで，

```
ON ERROR GOTO 10000
```

と入れて置きます。そして，エラー処理としては，計算不能であることを表示し，INPUT文に戻せばよいわけですから，エラー処理ルーチンは，

```
10000 IF ERR <> 11 THEN ON ERROR GOTO 0
10010 PRINT "ケイサン　フノウ":PLAY "O2C 3 RCR"
10020 RESUME 10
```

となります。10000行で行なっている処理は，Devision by zero 以外のエラーについてはエラー処理を行なわず，通常どおりエラーメッセージを出してストップするようにしたものです。ON ERROR文を使うと，どのようなエラーでも関係なくエラー処理ルーチンにジャンプしてしまいます。この場合エラー11に対してだけの対策しかしていませんから，11以外のエラーに対してRESUMEで10行に戻してもなんの意味もないためです。ON ERROR GOTO 0は，０行にジャンプするのではなく，エラー処理を行なわない，という意味を持っています。

RESUME文は，エラー処理ルーチンを実行した後の戻り先を指定するものですが，RESUMEだけだと，エラーの発生したステートメントへ，この例のように行番号，またはラベルをつけるとその行またはラベルに戻ります。RESUME NEXTとすると，エラーの発生した次のステートメントに戻りますから，マルチステートメントの途中へも戻ることができます。

実際の長いプログラムの中で，いくつもの場所で発生するエラーに対応するには，エラー処理ルーチンでERRとERLを組み合わせて，もっと複雑な処理をしなくてはならないのは言うま

らんだむめも

bitとbyte

ここに２冊の本があるとします。この２冊の本に書かれている情報の量を比べるにはどのような方法があるでしょうか。値段？　それともページ数？　文字数？　いずれにしても，本の情報量を比べるのは容易ではありません。

コンピュータの世界では，情報の量を表わすための単位があります。それが，bitやbyteです。

bitとは，binary digit（２進数字）の略で，情報量を表わす最小の単位です。どんな情報も，コンピュータの内部では，２進数に直して処理されます。情報を２進数に直したとき，２進数の桁数をbitで表現します。つまり，２進数の１桁が１bitにあたります。たとえば，10進数の37という数値を２進数に直すと，

37(10)＝100101(２)

となり，10進数37は６bitの情報というわけです。

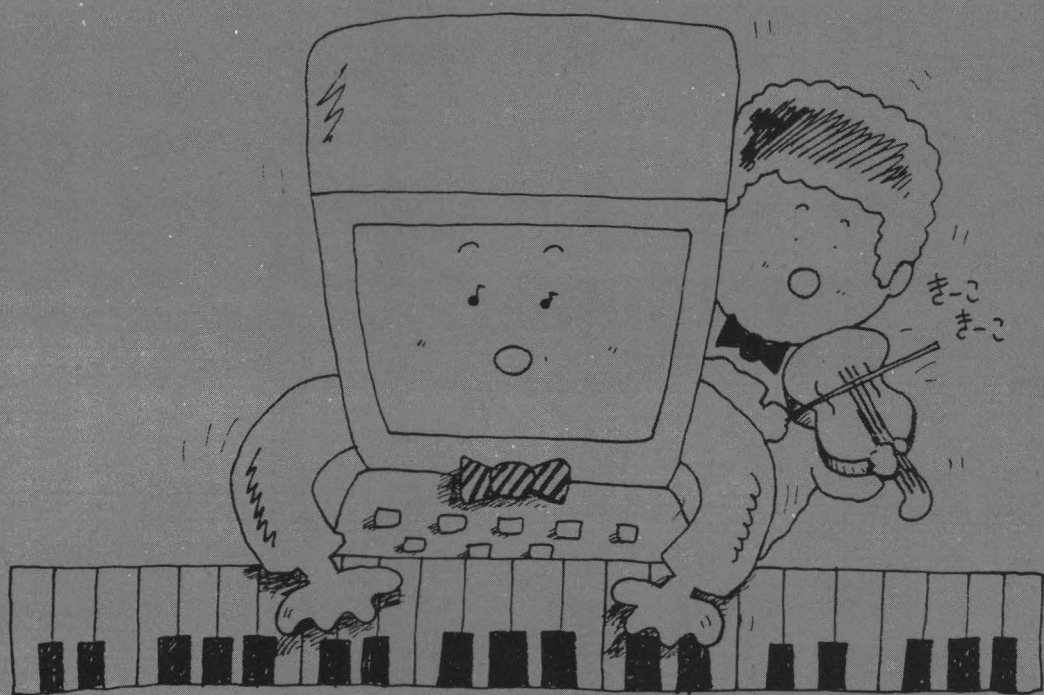
では，byteはどうでしょう。この言葉は，英和辞典を調べても載っていません。コンピュータの世界でしか使われない単位で，bitの集まりをいいます。８bitの情報を１つの単位として，byteで表わします。コンピュータが文字や記号などのキャラクタを１つ表現するには，１byte（＝８bit)が必要です。たとえば，"system"という言葉を表示するには，６文字，つまり６byteが必要となります，"system"は６byteの情報量といえるのです。

このように，byteという単位は，文字数を情報の量として表現するのにとても便利です。

でもありません。

エラー処理で特に注意しておきたいのは，ON ERROR文やエラー処理ルーチンはプログラムが一応完成して，プログラム自体が原因で起きるエラーを完全に直し，プログラムそのものではエラーを起こさないような修正などを完璧にしてから，データ入力など外的な原因によって起きるエラーだけを処理するようにすることです。最初からON ERROR文があると，Syntax errorなどでもエラー処理ルーチンにジャンプしてしまうため，RESUMEの戻り先などで不都合を生じることがあるのはもちろんですし，プログラムのバグを見つけることがむずかしくなります。このようなことから考えれば，ON ERRORで処理するエラーは，大変限られたものになります。あえて書くなら，よくできたプログラムでは，エラーコード11，27，29以外のエラーが発生することはほとんどありません。ON ERRORによるエラー処理はうまく使うと便利ですが，いいかげんに使うととんでもない処理が行なわれることになりますので，注意して使ってください。

7

X1の音楽才能は?

7-1 X1は名演奏家

●X 1の演奏機能

昔は非常に高価で扱いのむずかしかったシンセサイザを使った音楽も，最近ではすっかり身近なものになりました。安価で誰にでも扱えるシンセサイザがたくさん市販されています。

X 1には，プログラマブル・サウンド・ジェネレータ（PSG）という，ミニシンセサイザが内蔵されています。このPSGを使うとさまざまな音楽の演奏やゲームの効果音をつくることができます。ここでは，このPSGによる音作りの方法をみていきます。

X 1で音を出すには，楽譜を文字列にしてPLAY文によって演奏する方法とSOUND文を使って直接PSGのLSIにデータを送って音を出す2とおりの方法があります。ここではまず，PLAY文について見ていきましょう。

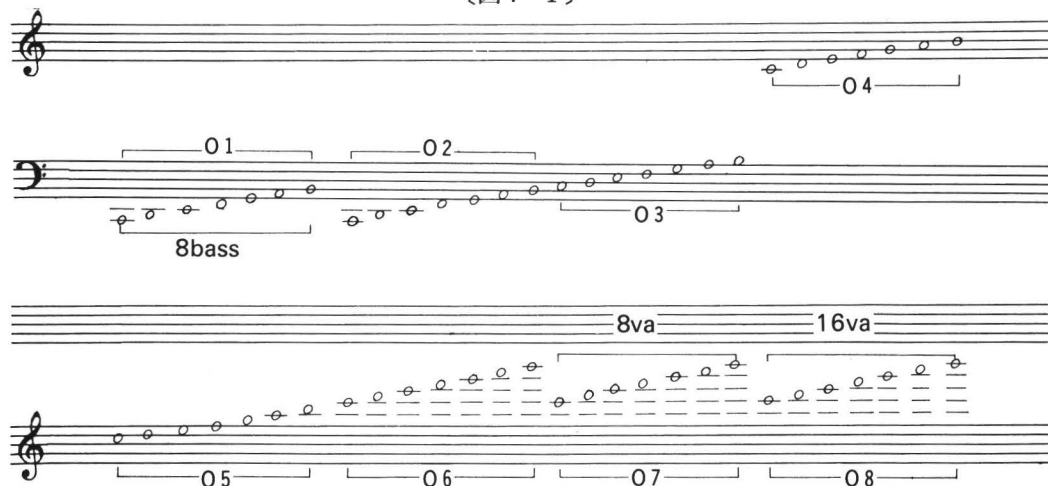
●PLAYを使えば……

音符を音にするには，音程，音量，音の長さの3つの要素を指定しなければなりません。

楽譜の読み方には，普通使われる「ドレミファ……」の他に，ドイツ読みの「CDEFGAH」や，英語読みの「CDEFGAB」がありますが，BASICでは英語読みを基本としたデータを使います。X 1で出せる音域は図7-1に示した8オクターブです。オーケストラの楽器でいえば，コントラバスより低い音からピッコロフルートより高い音まで出すことができます。さっそくダイレクトモードでPLAY文を使って音を出してみましょう。

```
PLAY "CDEFGAB"
```

ドレミファソラシ，と聞こえます。今度は半音を含めて8オクターブすべての音のプログラムを作ってみます（プログラム7-1）。

〔図7-1〕

〔プログラム7-1〕

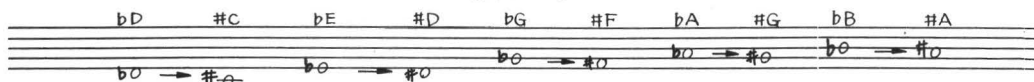
```
10 M$="C3#CD#DEF#FG#GA#AB"
20 FOR I=1 TO 8
30 MU$="O"+RIGHT$(STR$(I),1)+M$
40 PLAY MU$
50 NEXT
```

音を出すにはこのように，

PLAY　“文字列”　または，
MUSIC　“文字列”

とします。PLAYとMUSICは，後ろが文字列の場合，まったく同じ働きをします。わずかに異なるのは，PLAY文では文字列の中で，テンポの指定をできますが，MUSIC文にはこの機能がないという点だけです。MUSIC文では別にTENPO文によってテンポを指定しなければなりません。テンポとは，1分間に演奏される4分音符の数で，30〜7500まで指定することができます。何も指定しない場合は，テンポは120に設定されています。

実際の楽譜には，「#」（シャープ）や「♭」（フラット）があります。半音を指定する場合は，音程の前に「#」をつけます。「♭」はありませんのですべて「#」で代用します（図7-2）。たとえば「♭E」は「#D」に読みかえます。

〔図7-2〕

また，オクターブの指定は，「On」として，Oの後に図7-1の数値をつけます。オクターブ指定は，一度指定すると次に指定し直すまで変化しません。一部の音だけオクターブが変化する場合には，音名の前に「+」，「−」をつけることにより，On指定されたオクターブの上下を指定することができます。この「+」，「−」は重ねて使うことによって，その数だけ動かすこと（オフセット）ができます。図7-3の楽譜をPLAY文にすると次のようになります。

〔図7-3〕

```
PLAY "O4G6+D3+C5B5A5G3AG"
```

●音量は？

音程が指定できましたから，次は音量の指定をしましょう。音量は「Vn」を使います。VはVOLUMEの意味で，nには0から15までの数値を入れます。nが0ですと音は出ません。V15で音量は最大になります。

FOR〜NEXTを使ってこのnの値を変えながら音量を変化させてみましょう（プログラム7-2）。

〔プログラム7－2〕

```
10 FOR I=0 TO 15
20 M$="V"+RIGHT$(STR$(I),LEN(STR$(I))-1)+"A3"
30 PLAY M$
40 NEXT I
50 FOR I=15 TO 0 STEP -1
60 M$="V"+RIGHT$(STR$(I),LEN(STR$(I))-1)+"A3"
70 PLAY M$
80 NEXT I
```

また，V16とすると，後で説明するSOUND文と組み合わせることによって音色を変化させることができます。

●音の長さは？

次は音の長さの指定をします。音の長さは，表7－1のように4分音符を5とした相対的な値で指定します。休符はRで指定し，その長さは音符を決める数値と同じです。

表7－1

音の長さ		対応する整数
32分音符	$\frac{1}{8}$	0
16分音符	$\frac{1}{4}$	1
付点16分音符	$\frac{3}{8}$	2
8分音符	$\frac{1}{2}$	3
付点8分音符	$\frac{3}{4}$	4
4分音符	1	5
付点4分音符	$1\frac{1}{2}$	6
2分音符	2	7
付点2分音符	3	8
全音符	4	9

音の長さは，一度指定すると，オクターブの指定のように，指定し直すまで変化しません。

これで，音符を音にすることができるようになりました。これに対し，「＋」「－」「#」は後に書く1音だけに作用します。

音楽には，「メロディー」，「リズム」，「ハーモニー」の3つの要素があります。音程，音量，音の長さを指定することができるようになりましたから，これで，「メロディー」と「リズム」という2つの要素を自由に操れるようになったわけです。残るは「ハーモニー」です。

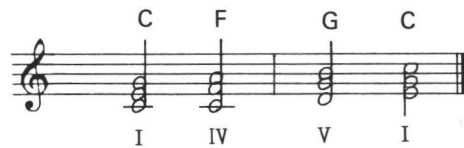
さて「ハーモニー」ですが，X1では3つの音を同時に出すことができます。つまり，三重和音を使うことができるのです。PLAYの後に，

PLAY "音1：音2：音3"

のように，「：」(コロン)で区切って文字列を書くと，この3つの音が重なって演奏されます。ハ長調の3和音（図7－4）を演奏するのが次のプログラムです。

〔プログラム7－3〕

```
10 PLAY "O4V15:O4V15:O4V15"
20 PLAY "C7CDE:E7FGG:G7AB+C"
```

〔図7－4〕

ここでは，10行で3音のオクターブと音量を決めておき，20行で音程だけを指定するという方法を使っています。

次からは，実際に楽譜から曲を演奏させるいろいろなテクニックを見ていきましょう。

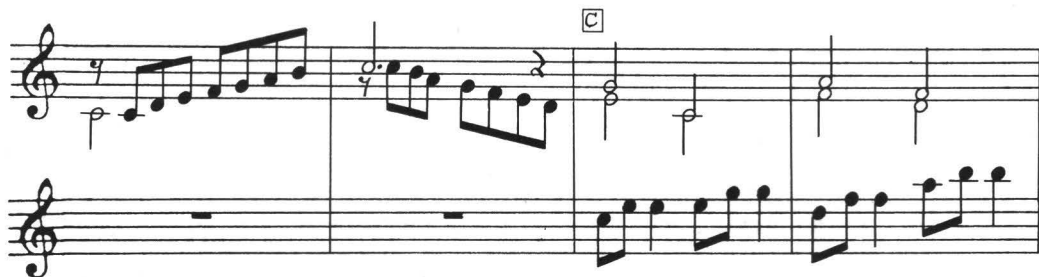
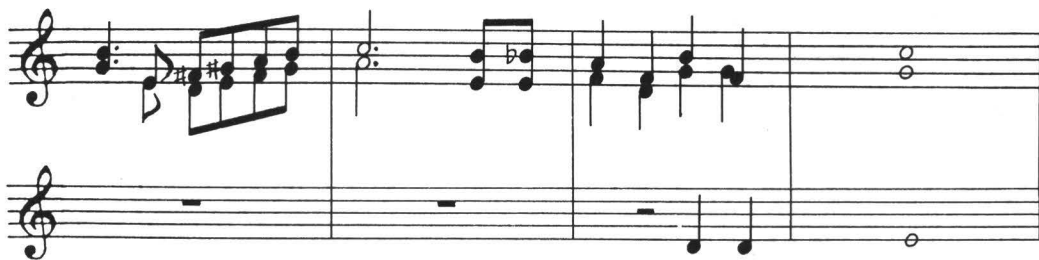
7-2 ドレミの歌を演奏してみよう

●ドレミの歌，歌えるかな

さて，いよいよPSGを活用したＸ１に音楽を演奏させてみましょう。テーマとして，映画「サウンド・オブ・ミュージック」のドレミの歌を取りあげてみました。まず最初に，全体の楽譜を見てみましょう（図７－５)。

楽譜は苦手だという人も心配することはありません。楽譜をそのまま文字列のデータに直せばよいのですから。

メインルーチンは，READ～DATAを使い，曲の終わりに「FINE」と書いておき，これが読みとれた時にプログラムが終了するようにします。それ以外のときは，読み出したデータをPLAY文で演奏し，GOTO文でREADに戻すようにしてあります。

〔図７－５〕

●休符をつける

まず，4小節目まで演奏させてみます。

〔プログラム7-4〕

```
90 TEMPO 144
100 READ M$
110 IF M$="FINE" THEN END
120 PLAY M$
130 GOTO 100
140 DATA "C6D3E6C3E5CE7"
150 DATA "D6E3FFEDF7"
160 DATA "FINE"
```

これを走らせて音を聞いてみると，全部の音がつながって，なんとなくだらしのないメロディーになってしまいます。特に，3小節目は8分音符がまったくわからなくなってしまいます。

実際，楽器で演奏したり，歌ったりする場合，必ず音と音を区切っています。たとえば3小節目は図7-5Bのように演奏すると考えることができます。この考え方でデータをつくり直したのが次のプログラムです（プログラム7-5）。

〔プログラム7-5〕

```
90 TEMPO 144
100 READ M$
110 IF M$="FINE" THEN END
120 PLAY M$
130 GOTO 100
140 DATA "V15O4C5R3D1RE5R3C1RE4R1C4R1E6R3"
150 DATA "D5R3E1RFRFRERDRF8R5"
160 DATA "FINE"
```

今度はリズムもはっきりして音楽らしく聞こえます。

休符をつけるということは，1つの音符を音の部分と休みの部分に分割するという意味です。図7-6に音符の分割の例をあげておきます。楽譜上でスラーやタイなどの音符をつないで演奏する記号がついている場合以外は，この方法で休符をつけるようにします。音符には細かな表情があり，それをコンピュータで美しく演奏するには少々コツがいります。あくまで聞いた感じが大切ですから必ずしもこのように分割するわけではありません。何度も聞いてみて直していくことも必要です。

〔図7-6〕 4分音符、

普通の音　スタッカート　テヌート

この音を休符にする

F5　F4R1　F3R　F3F2R0

F1R　F0 R2　F2 R0

●のばす音の場合（曲の感じにより使いわける）

F8F3R　または　F8 R5

●パッセージの場合（主旋律の場合）

C1 RD RE RF RG RA BR+CR

●付点音符の場合

F5R3F1R　または　F5 F1 R FR

「ドレミの歌」の全曲のデータを作ったプログラムをあげておきます。Ⓐの部分は1声，Ⓑの部分は2声になりハーモニーをつけてあります。Ⓒの部分からは3声になり，最上音の旋律は，音を分割しないでなめらかにつないで，下の段のリズムを目立たせるように作ってみまし

た（プログラム7-6）。

音符の分割など，少々面倒ですが，慣れると比較的簡単です。他の曲のデータも作って演奏させてみるとよいでしょう。

〔プログラム7-6〕

```
10 ' ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
20 ' ■                                    ■
30 ' ■             ﾄﾞﾚﾐ ﾉ ｳﾀ              ■
40 ' ■                                    ■
50 ' ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
60  '
90 TEMPO 144
100 READ M$
110 IF M$="FINE" THEN END
120 PLAY M$
130 GOTO 100
140 DATA "V1504C5R3D1RE5R3C1RE4R1C4R1E6R3:V1504:V1504"
150 DATA "D5R3E1RFRFRERDRF8R5"
160 DATA "E5R3F1RG5R3E1RG4R1E4R1G6R3:C5R3D1RE5R3C1RE4R1C4R1E6R3"
170 DATA "F5R3G1RARARGRFRA8R5:D5R3E1RFRFRERDRF8R5"
180 DATA "G5R3C1RDRERFRGRA8R5:E5R3C1R-#ARCRDRERF8R5"
190 DATA "A5R3D1RER#FRGRARB8R5:#F5R3D1RERDRER#FRG8R5"
200 DATA "B5R3E1R#FR#GRARBR+C7+C3RB1R#AR:#G5R3E1RDRER#FR#GRA7A3R
E1RER"
210 DATA "A4R1F4R1B4R1G4R1+C8+C3R:F4R1D4R1G4R1F4R1G8G3R:R5R5D4R1
D4R1E8E3R"
220 DATA "R3C1RDRERFRGRARBR+C8R5:C8R5R3+C1RBRARGRFRERDR"
230 DATA "G7CAF:E7CFD:O5C1RERE4R1E1RGRG4R1D1RFRF4R1A1RBRB4R1"
240 DATA "ECD9:CCR9:C1RERE4R1E1RGRG4R1D1RFRF4R1A1RBRB4R1"
250 DATA "G7CAB:E7CFD:C1RERE4R1E1RGRG4R1D1RFRF4R1A1RBRB4R1"
260 DATA "+C+DE9:EFG9:R3+C1RBRARGRFRERDRC9"
270 DATA "FINE"
```

らんだむめも

REPEAT ON/OFF

キー入力のリピート機能（1つのキーを押し続けると同じ文字が続けて入力される）は，大変便利なものですが，ときにはこの機能がかえってわずらわしく感じられることもあります。たとえば，プログラムが入力待ちの状態で，1文字だけ打ち込んだつもりなのに続けて2文字が入ってしまった，なんてよくあることですね。

こんなときは，REPEAT OFFと命令してやるとリピート機能を止めることができます。もちろん，プログラム中に書いてもかまいません。また，元の状態に戻すにはREPEAT ONと命令します。

7-3 PSGと直接話してみよう

●音の秘密は周波数

PLAY文を使えば，簡単に曲を演奏させることができました。ところで，この演奏された音はどのようにして作られているのでしょうか。

コンピュータは，電気の0と1の記号によって動いています。この記号をパルスといいます。パルスには基準となる決められた周波数があります。X1の場合，4MHz（1秒間に400万回の周期）になっています。この基準となる周波数をクロック周波数と呼び，計算のスピードなどコンピュータの動作の速さを決める要素になっています。この周波数をある割合で分割していくと，周波数がしだいに小さくなっていきます。たとえば，9090分の1にすると，440Hzになります。これは，「O4A」の音の周波数にあたります。

このクロック周波数を分割する操作（分周）を行なうのが，PSGのLSIです。音は，1オクターブ上がると周波数が倍になっており，その間を12に分けると半音階が得られます。O2CからO7Cまでの5オクターブの半音階の各音の周波数を表7-2にあげてみました。

このPSGには，音を出すために必要なデータを送りこむところ（レジスタ）が14カ所あります。レジスタとは，記憶されたデータを特定の機能によって演算を行なうものです。PCGの各レジスタの働きは，次の表7-3のようになっています。

表7-2

音　階	周波数	音　階	周波数
O2C	65.40	G	392.00
#C	69.30	#G	415.30
D	73.42	A	**440.00**
#D	77.78	#A	466.16
E	82.41	B	493.88
F	87.30	O5C	523.25
#F	92.50	#C	554.37
G	98.00	D	587.33
#G	103.83	#D	622.25
A	110.00	E	659.26
#A	116.54	F	698.46
B	123.47	#F	739.99
O3C	130.81	G	783.99
#C	138.59	#G	830.61
D	146.83	A	880.00
#D	155.56	#A	932.33
E	164.81	B	987.77
F	174.61	O6C	1046.50
#F	185.00	#C	1108.70
G	196.00	D	1174.70
#G	207.65	#D	1244.50
A	220.00	E	1318.50
#A	233.08	F	1396.50
B	246.94	#F	1480.00
O4C	261.63	G	1568.00
#C	277.18	#G	1661.20
D	293.66	A	1760.00
#D	311.13	#A	1864.70
E	329.63	B	1975.70
F	349.23	O7C	2093.00
#F	369.99		

表7-3

レジスタ番号	レジスタの働き
0	チャンネルAの周波数
1	〃
2	チャンネルBの　〃
3	〃
4	チャンネルCの　〃
5	〃
6	ノイズの周波数
7	各チャンネルの音色（ノイズトーン）
8	チャンネルAの音量
9	チャンネルBの音量
10	チャンネルCの音量
11	音の波形
12	〃
13	音の波形の周期

このレジスタにデータを送りこむための命令がSOUND文です。

●SOUND文を使って

まずこのSOUND文を使って440Hzの音（O4A）を出してみましょう。

とりあえず必要なレジスタは，チャンネルAの音程を決めるレジスタ0とレジスタ1，チャンネルAから音を出すように決めるレジスタ7，チャンネルAの音量を決めるレジスタ8の4つです。チャンネルは，PLAY文で三重和音を出したときのそれぞれの音に対応しています。つまり，PLAY "A：B：C" は，チャンネルA，B，Cから同時に音が出て，和音となっていたのです。

音程を決めるには，分周比というものを計算しなければなりません。PSGの中では，あらかじめクロック周波数が32分の1に分周されており，簡単に分周比を求めることができます。たとえば，440Hz（O4A）の時の分周比は4000000／440／32≒284となります。次にこの分周比をレジスタに送りこむデータに直します。次の〔プログラム7-7〕は，出したい音の周波数からレジスタ0，1に送るデータを求めるものです。

〔プログラム7-7〕

```
10 INPUT "ｼｭｳﾊｽｳ ";FRQ
20 P=4*10^6/32/FRQ
30 REG1=P¥256
40 REG0=INT(P-REG1*256)
50 PRINT REG0,REG1
```

PSGに入ってくるのは，クロック周波数を2分の1にしたもので，さらに内部で16分の1に分周しています。

30行では「¥」が使われています。これは「＋」「－」「＊」などの算術演算子の1つで，整数同士の割り算をして，その答を整数で与えます。440を入れてみると，REG 0 が28，REG 1 が1と表示されます。SOUND文のオペランドに，この値を入れてデータを送ってみます。

SOUND 0, 28：SOUND 1, 1↵

まだ音は出ません。どのチャンネルから音を出すか決めていないためです。レジスタ7にデータを送ります。送るデータは次のようになります。

ノイズ			トーン			
C	B	A	C	B	A	チャンネル
32	16	8	4	2	1	全部加えると63になる，出したい音のところの数値を63から引く。

この場合はチャンネルAからも音程の決まったきれいな音（トーン）を出すのですから，63－1＝62をレジスタ7に送ります。チャンネルB，チャンネルCからトーンを出す場合には，それぞれ61，59をレジスタ7に送ります。

SOUND 7, 62↵

としてみます。これで音が出ない場合は，チャンネルAの音量が小さいことが考えられますから，レジスタ8にもデータを送ります。このデータは，PLAY文の音量のパラメータと同じで，0から15まででです。

SOUND 8, 15↵

これで音が出たはずです。SOUND文の場合，一度データを送ると，音を止めるデータを送る

までその音が鳴り続けます。音を止めるには，SOUND 8, 0(チャンネルAの音量を0にする)または，SOUND7, 63(すべてのチャンネルの音を止める)と命令します。あるいは，CTRL＋D キーを押して止めることもできます。

表7-2の各周波数に対応するレジスタ0，1のデータを計算してみたのが次の〔プログラム7-8〕です。

〔プログラム7-8〕

```
100 READ M$,FRQ
110 IF M$="END" THEN SOUND7,63:END
120 P=4*10^6/FRQ/32
130 REG1=P¥256
140 REG0=INT(P-REG1*256)
145 SOUND0,REG0:SOUND1,REG1:SOUND7,62:SOUND8,15
150 IF LEFT$(M$,1)<>"O" THEN PRINT" ";
160 PRINT M$,REG0,REG1
170 GOTO 100
1000 DATA O2C,65.4,#C,69.3,D,73.42,#D,77.78,E,82.41,F,87.30,#F,9
2.5,G,98,#G,103.83,A,110,#A,116.54,B,123.47
1010 DATA O3C,130.81,#C,138.59,D,146.83,#D,155.56,E,164.81,F,174
.61,#F,185,G,196,#G,207.65,A,220,#A,233.08,B,246.94
1020 DATA O4C,261.63,#C,277.18,D,293.66,#D,311.13,E,329.63,F,349
.23,#F,369.99,G,392,#G,415.3,A,440,#A,466.16,B,493.88
1030 DATA O5C,523.25,#C,554.37,D,587.33,#D,622.25,E,659.26,F,698
.46,#F,739.99,G,783.99,#G,830.61,A,880,#A,932.33,B,987.77
1040 DATA O6C,1046.5,#C,1108.7,D,1174.7,#D,1244.5,E,1318.5,F,139
6.5,#F,1480,G,1568,#G,1661.2,A,1760,#A,1864.7,B,1975.7
1050 DATA O7C,2093,END,0
```

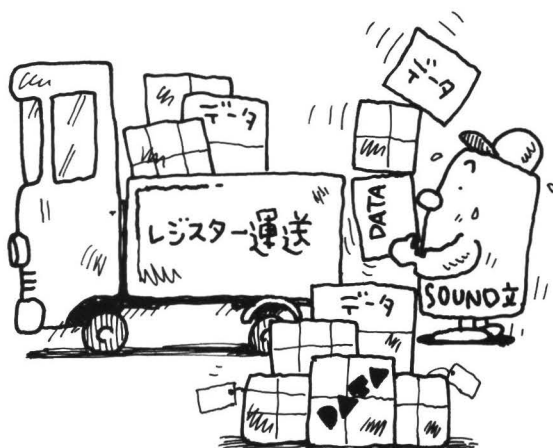
DATA文に表7-2の音名と周波数を置き，READ文で変数M＄とFRQのそれぞれにデータとして読み込んで計算しています。そしてこの値をレジスタ0と1に送って実際の音を出しています。

●チャンネルから音を出すには

チャンネルB，Cから音を出すのもレジスタ2，3と4，5に送るデータは同じで，レジスタ7とレジスタ9，10の値が変わるだけです。レジスタ7に送る値は，2進数を使うとわかりやすくなります（図7-7)。

〔図7-7〕

それぞれのチャンネルで，音を出すのが0，出さなければ1と指示をしています。この各チャンネルの指定を1つにまとめると「111001」という2進数の文字列になります。先頭に「&B」をつけて0と1を並べると，2進数として扱われます。16進数の先頭に「&H」をつけるのと同じです。

先ほどのデータのうち，音を出したい

ところを0，出さないところを1として並べ，6桁の2進数にします。たとえば，チャンネルCからノイズを，チャンネルA，Bからトーンを出す場合，

```
011100
↑   ↑↑――チャンネルA，トーンON
│   │└――チャンネルB，トーンON
└――――――――チャンネルC，トーンON
```

という形になります。これをそのまま2進数としてレジスタ7に送ってやればよいわけです。

```
SOUND 7, &B011100
```

とすることになります。

●レジスタの変化で効果音を

音程を決めるレジスターに送る値を連続的に変化させると，PLAY文とは違った効果を出すことができます（プログラム7-9）。

〔プログラム7-9〕

```
10 ' UFO
20 FOR K=1 TO 5
30 FOR I=128 TO 60 STEP-1
40 SOUND0,I:SOUND1,0:SOUND7,62:SOUND8,15
50 NEXT:NEXT
60 SOUND7,63
```

ここでは，レジスタ0に送る値を0.5のステップにしていきますが，実際はレジスタにセットされる値は整数だけです。小数点以下の数値は，レジスタに受けつけられるときには四捨五入されて整数に直されています。〔プログラム7-10〕で0.5をステップにしているのは，単に変化の速度を調整するためです。またノイズも組み合わせて使っています。これらのレジスタの使い方を次のところで調べてみます。

〔プログラム7-10〕

```
10 ' ツイラク
20 FOR I=0 TO 255 STEP .5
30 SOUND0,I:SOUND1,0:SOUND7,62:SOUND8,12
40 NEXT
50 SOUND6,31:SOUND7,55:SOUND8,16:SOUND11,10:SOUND12,60:SOUND13,0
```

7-4 PSG完全マスターのために

●音を変化させるには

〔プログラム7-10〕の墜落の音について，もう一度調べてみましょう。

40行まででFOR～NEXTによって変化させた値をレジスタ0に送って，高い音から低い音へ連続的に変化させています。最後に，爆発音のような音が出ました。50行を見てみると，レジスタ6，11，12，13と，今まで使っていなかったレジスタが登場してきました。

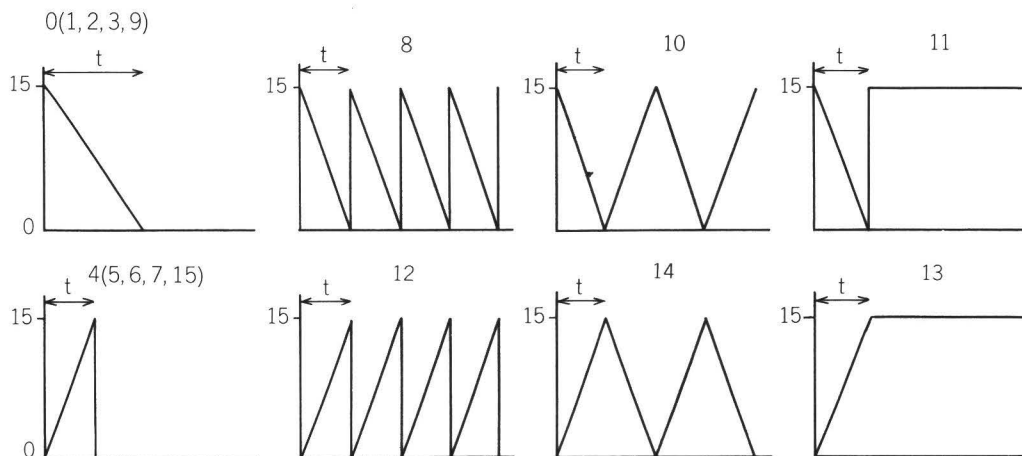
トーンを出すには，レジスタ0から5までの3組のレジスタにデータを送っていました。この場合のように雑音に近い音（ノイズ）を出すには，レジスタ6にデータを送ります。レジスタ6へ送ることのできる値は，1から31です。この数値が大きいと，低い感じの「ザァー」という音で，小さくすると，高い感じの「シャー」という音になります（プログラム7-11)。

〔プログラム7-11〕

```
10 CLS
20 FOR I=1 TO 31
30 SOUND 6,I:SOUND 9,15:SOUND 7,47
40 LOCATE 0,12:PRINT "Register 6=";I
50 PAUSE 5
60 NEXT I
70 SOUND 7,63
```

ノイズを出すには，その他にレジスタ7でどのチャンネルから音を出すか決め，8～10で音量も決めなければなりません。レジスタ7のデータによっては，ノイズとトーンを1つのチャンネルから重ねて出すこともできます。

レジスタ11～13を使うと，音を変化させることができます。音の変化の形（エンベロープ波形）には，図7-8に示した8種類があります。

この図の上にある数値がレジスタ13に送るデータで，Tで示した時間がレジスタ11と12にセットされるデータで変化する単位です。

変化の時間Tは，音量が0から最大15まで変化する時間で，約8秒から1000分の1秒まで決められます。〔プログラム7-12〕は，Tの値(秒)に対応するレジスタ12，13の値を計算するものです。

〔プログラム7-12〕

```
10 INPUT "T=",T
20 B=1/T*2
30 X=4*10^6/(256*B)
40 REG12=INT(X/256)
50 REG11=INT(X-REG12*256)
60 SOUND 0,28:SOUND 1,1:SOUND 7,62:SOUND 8,16
70 SOUND 11,REG11:SOUND 12,REG12:SOUND 13,&B1000
80 PRINT REG11,REG12
```

Tに対応するレジスタ11と12の値を，表7-4に示しておきます。

表7-4

T	レジスタ　11	レジスタ　12
T＝8	REG11＝36	REG12＝244
T＝7	REG11＝159	REG12＝213
T＝6	REG11＝27	REG12＝183
T＝5	REG11＝150	REG12＝152
T＝4	REG11＝18	REG12＝122
T＝3	REG11＝141	REG12＝91
T＝2	REG11＝9	REG12＝61
T＝1	REG11＝132	REG12＝30
T＝.9	REG11＝119	REG12＝27
T＝.8	REG11＝105	REG12＝24
T＝.7	REG11＝92	REG12＝21
T＝.6	REG11＝79	REG12＝18
T＝.5	REG11＝66	REG12＝15
T＝.4	REG11＝52	REG12＝12
T＝.3	REG11＝39	REG12＝9
T＝.2	REG11＝26	REG12＝6
T＝.1	REG11＝13	REG12＝3
T＝.09	REG11＝191	REG12＝2
T＝.08	REG11＝112	REG12＝2
T＝.07	REG11＝34	REG12＝2
T＝.06	REG11＝212	REG12＝1
T＝.05	REG11＝134	REG12＝1
T＝.04	REG11＝56	REG12＝1
T＝.03	REG11＝234	REG12＝0
T＝.02	REG11＝156	REG12＝0
T＝.01	REG11＝78	REG12＝0
T＝.009	REG11＝70	REG12＝0
T＝.008	REG11＝62	REG12＝0
T＝.007	REG11＝54	REG12＝0
T＝.006	REG11＝46	REG12＝0
T＝.005	REG11＝39	REG12＝0
T＝.004	REG11＝31	REG12＝0
T＝.003	REG11＝23	REG12＝0
T＝.002	REG11＝15	REG12＝0
T＝.001	REG11＝7	REG12＝0

ここでさらに，各レジスタの働きとデータの範囲を表7-5にまとめておきましょう。

ノイズやエンベロープを使い，音の出かたをうまく組み合わせていくと，かなり豊富な音を

表7－5

レジスター	機　　　能	データ範囲
0	チャンネルAのトーンの音程(細かく変化させる)	0～255
1	〃　　　　(粗く　　〃　　)	0～15
2	チャンネルBのトーンの音程(細かく　〃　　)	0～255
3	〃　　　　(粗く　　〃　　)	0～15
4	チャンネルCのトーンの音程(細かく　〃　　)	0～255
5	〃　　　　(粗く　　〃　　)	0～15
6	ノイズの高さ	0～31
7	チャンネルA，B，C，に対するトーン，ノイズのオン・オフ。	0～63*
8	チャンネルAの音量(16ならエンベロープ)	0～15，16
9	〃　B　〃　(　　〃　　)	0～15，16
10	〃　C　〃　(　　〃　　)	0～15，16
11	エンベロープの速さ(細かく変える)	0～255
12	〃　　(粗く　〃　)	0～255
13	エンベロープの形状	0～15

＊2進数6桁で表示でき、ONなら0，OFFなら1。
この2進数を10進数に直したものがデータとなる。

```
   ノイズ トーン
&B 0 0 0 0 0 1
   C B A C B A チャンネル
```

出すことができます。一見複雑そうですが，1つ1つのレジスタの働きをよく考えて組み合わせていけば，ゲームの効果音などに大変便利に使えます。

●ゲームの効果音も自由自在

たとえば，PLAY文で文字列データを演奏する場合，エンベロープを変化させると，また違った感じに聞こえてきます。そのためには，音量を指定するVを16にセットします。こうしておくと，レジスタ11，12，13で決めたエンベロープ形状で音楽を演奏させることができます。〔プログラム7-13〕では，これを利用して，エレクトーンとピアノに似た音階を演奏します。

〔プログラム7-13〕

```
10 TEMPO 144
20 M$="V1604C5DEFGAB+C+CBAGFEDC8R5"
30 SOUND 11,255:SOUND 12,1:SOUND 13,&B1110
40 PLAY M$
50 SOUND 11,30:SOUND 12,30:SOUND 13,0
60 PLAY M$
70 PLAY "V15C5DEFGAB+C+CBAGFEDC8"
```

PLAY文で音を続けてならすと，音がつながってしまうことは説明しましたが，レジスタ13でエンベロープ形状を0（1，2，3，9も同じ）にし，エンベロープの変化時間をうまく指定すると，音がつながってしまうことがさけられます。特に，ピアノ曲やギター曲をPLAY文で演奏するときに有効に使うことができます。最後に普通の形式で演奏しますから違いを聞きわけてください。

レジスタ6で出すノイズも同じように，PLAY文と組み合わせることができます。このときに，エンベロープ形状を0にすると，ドラム（小太鼓）のような音を出すことができます。〔プログラム7-14〕はこれを使ったドラムソロです。

〔プログラム7-14〕

```
10 TEMPO 120
20 SOUND10,16:SOUND11,20:SOUND12,10:SOUND6,1:SOUND7,63-32
```

```
30 READ M$
40 IF M$="Fine" THEN END
50 PLAY M$
60 GOTO 30
70 DATA "R9:R9:O1V16C3CCC1CC3CCC1CC3CC1CCCC3CCC1C"
80 DATA "R9:R9:C3CCC1CC3CCC1CCCCCCCCCC3CR3R"
90 DATA "R9:R9:C3CR5C3CR5C0CCCCCCCCCCCCCCCC3CR"
100 DATA "Fine"
```

30行でエンベロープ形状と時間，ノイズの高さを決めて，レジスタ7によって，チャンネルCからノイズだけを出すように指定しています。PLAY文では，必ずトーンも出てしまいますので，オクターブを1にして目立たないようにしています。データには休符を入れてませんが，音の長さにかかわらずエンベロープによる音の減衰が優先されるため，適当に区切られます。また，この形状は，音の立ち上がりが鋭いので，音の頭の部分（アタック）がはっきりします。

●多彩な音作り

X 1のPSGで音楽を演奏する場合，PLAY文のところで説明した音符の分割の方法と，ここで説明したSOUND文によるエンベロープ形状の変化の組み合わせで，多彩な音作りができます。たとえば，チャンネルAから普通の方法でメロディーを出し，チャンネルBからエンベロープをかけたピアノ風の音を出して伴奏するということもできます。〔プログラム 7-15〕では，ビゼーの「アルルの女」の有名なフルートソロの頭の部分をこの方法で演奏してみました。

〔プログラム 7-15〕

```
10 TEMPO 70
20 SOUND11,12:SOUND12,30:SOUND13,0
30 READ M$
40 IF M$="Fine" THEN END
50 PLAY M$
60 GOTO 30
100 DATA "V15R8:V16O3C3G+CG+EG"
110 DATA "R8:V16O3C3G+CG+EG"
120 DATA "O5E4E0RE4D1CDEF:C1RGR+CRGR+ERGR"
130 DATA "G3E+CG+E5:C3G+CG+EG"
140 DATA "D4D0RD4E1DC-B-A:D1RAR+DRARFRAR"
150 DATA "-G3-BD-BG5:-G3GBG+G8"
200 DATA "Fine"
```

それでは最後に，今までにでてきたさまざまなテクニックを駆使したオリジナル曲，「デジタル・ブルースカイ」を紹介します（プログラム 7-16）。

この曲は，3部形式の繰り返しがあるため，曲を4つの部分にわけてラベルをつけます。そして，はじめにこのラベルの演奏順序をFOR～NEXTで読み込んでおき，これにしたがって演奏していきます。

軽快なマーチです。楽しんでください。

〔プログラム7-16〕

```
10 ' ┌─────────────────────────────────┐
20 ' |        Digital Blue Sky         |
30 ' |                                 |
40 ' |      Original Music Data        |
50 ' |                                 |
60 ' |          by HAL.Sugawa          |
70 ' |                                 |
80 ' | ♣ STRATFORD COMPUTER CENTER ♣ |
90 ' └─────────────────────────────────┘
100 TEMPO 144
110 WIDTH80:SCREEN0,0:COLOR7,1:CLS:FE=0
120 SOUND11,20:SOUND12,10:SOUND13,0:SOUND6,1:SOUND7,63-32
130 RESTORE540:FOR I=1 TO 7:READ RES$(I):NEXT
140 FOR I=1 TO 7
150 IF I>1 THEN SOUND7,56
160 IF I=5 THEN SOUND11,20:SOUND12,10:SOUND13,0
170 RESTORE RES$(I)
180 READ M$
190 IF M$="|" THEN GOTO220
200 PLAY M$
210 GOTO 180
220 NEXT
230 LABEL"DrSOLO"
240 DATA "R9:R9:O1V16C3CCC1CC3CCC1CC3CC1CCCC3CCC1C"
250 DATA "R9:R9:O1V16C3CCC1CC3CCC1CCCCCCCCCC3CR3R"
260 DATA "R9:R9:C3CR5C3CR5C0CCCCCCCCCCCCCCCC3CR"
270 DATA "|"
280 LABEL"INTRO"
290 DATA "V15O5C1RC0RCRC1R-G0RCRE1RE0RERE3R:V15O4G1RG0RGRG1RE0RG
RG1RG0RGRG3R:V15"
300 DATA "E1RE0RERE1RC0RERG1RG0RGRG3R:G1RG0RGRG1RG0RGR+C1R+C0R+C
R+C3R"
310 DATA "G1RG0RGRG1RERG1RG0RGRG1RER:O5C1RC0RCRC1R-GRC1RC0RCRC1R
-G0R"
320 DATA "G1RG0R#F0RG1RARG3R:D1RD0RDRD1RDRD3R"
330 DATA "|"
340 LABEL"THEME"
350 DATA "O4G3R1GO5E4#D1E5R3-G1RCRER:V13O2G3R1G+C3RGRCRC1RGR:V13
R5O6C1RC-BC1RC1-BC3"
360 DATA "G1RGRFRERF3R-G3R1-G:C3RGRCRG3R1G:O6C1RC-BC1RC1-BC3"
370 DATA "D4#C1D5R3-G1R-BRDR:B3RGRBRC1RGR:D1RDCD1RDCD3"
380 DATA "F1RFRER#DRE3R-G3R1-G:B3RGRBRG3R1G:F1RFEF1RFEF3"
390 DATA "G4#F1G5R3C1RERGR:C3RGRCRC1RGR"
400 DATA "#AR#ARAR#GRA1R4G4R1:A3RFRARG1RBR"
410 DATA "F4E1F1R-GR-BRERD4R1D4R1:B3RGRBRGG1RARBR"
420 DATA "C7C3R:C3RGRCR:C1-C-D-E-F-G-A-BC"
430 DATA "|"
440 LABEL"TRIO"
450 DATA "C4R1C7C2R0-A3CF:V16O4R5R3C1RCRCRR3C1RCRCR:V16O4R5R3E1R
ERERR3E1RERER
460 DATA "A7F4R1F4R1:R3C1RCRCRR3C1RCRCR:R3F1RFRFRR3F1RFRFR"
470 DATA "F5F2R0E3GFC3R1-B:R3-#A1R-#AR-#ARR3-#A1R-#AR-#AR:R3D1RD
RDRR3D1RDRDR"
480 DATA "-#A7-#A4R1-#A4R1:R3-#A1R-#AR-#AR-#A3RR5:R3D1RDRDRD3R5"
490 DATA "-#A5-#A4R1-#A3-A-#AD:R3-#A1R-#AR-#ARR3-#A1R-#AR-#AR:R3
D1RDRDRR3D1RDRDR"
500 DATA "G5G4R1D4R1D4R1:R3C1RCRCRR3-#A1R-#AR-#AR:G3G1RGRGRR3D1R
DRDR"
510 DATA "F5F4R1E1RFRGRAR:R3C1RCRCRR3C1RCRCR:R3F1RFRFRR3G1RGRGR"
520 DATA "F7R4:F1RF0RFRF1RFRF3R:C1RC0RCRC1RCRC3R"
530 DATA "|"
540 DATA "DrSOLO","INTRO","THEME","THEME","TRIO","INTRO","THEME"
```

7-5 X1を電子オルガンに

●キーを鍵盤に！

今まで，PLAY文やSOUND文で，プログラムによって音を出すことを調べてきました。今度は，X１のキーボードを文字どおりピアノやオルガンなどのキーボード鍵盤と同じように使って音を出すプログラムを考えてみましょう。X１を電子オルガンにしようというわけです。

楽器とするには，たとえば[F]のキーが押されている間はずっとそれに対応する音が出るようにしなければなりません。このようなときは，INKEY$(0)を使うとよさそうです。とりあえず，キーボードのF, G, H, J, Kを「ド」から「ソ」までの音に対応させることを考えてみます。

INKEY$(0)を使うと，キーが押されている間は，そのキーの文字が変数に代入されて戻ってきます。これを利用して作ったのが次の〔プログラム７-17〕です。キーの判断は大文字を使いましたので，[CAPS LOCK]キーを押したまま使います。

〔プログラム７-17〕

```
10 TEMPO 120
20 Z$=INKEY$(0)
30 IF Z$="F" THEN PLAY "O4C0"
40 IF Z$="G" THEN PLAY "O4D0"
50 IF Z$="H" THEN PLAY "O4E0"
60 IF Z$="J" THEN PLAY "O4F0"
70 IF Z$="K" THEN PLAY "O4G0"
80 GOTO10
```

どうやら音を出すことはできましたが，音が細かく途切れてしまい，あまり音楽的とはいえません。これは，PLAY文によってINKEY$（0）に１つの文字が入ってくるたびに，音の長さが0，つまり32分音符を演奏させているためです。普通PLAY文で32分音符を演奏される場合は，音がつながって聞こえるのですが，この場合は，１つの音符に

〔図７-９〕

表７-６

音　名	レジスター０	レジスター１
O４C	221	1
〃　D	169	1
〃　E	123	1
〃　F	101	1
G	62	1

INKEY$(0)の処理やIF文による判断がついているため，音が途切れてしまうのです。というのは，実際に音が出ているのは30行から70行の後半のPLAY文の部分だけで，ほかの部分を実行している間は音が出ないからです。

そこでSOUND文を使うことにします。SOUND文ならば，一度データをレジスタに送っておけば，次のデータが送られないかぎり，その音が鳴りつづけます。各音のレジスタ0と1に送るデータの値は，前に求めることができました。O4CからGまでのデータの値は表7-6のようになります。

これを利用して作りかえたのが，次の〔プログラム7-18〕です。

今度はうまく音を出すことができました。そこで，キーボードのキーに図7-9のように音を割りつけ，音域を広げてみることにします。

〔プログラム7-18〕

```
10 TEMPO 120
20 Z$=INKEY$(0):IF Z$="" THEN SOUND 7,63:GOTO 20
30 IF Z$="F" THEN R0=221:R1=1:GOTO 90
40 IF Z$="G" THEN R0=169:R1=1:GOTO 90
50 IF Z$="H" THEN R0=123:R1=1:GOTO 90
60 IF Z$="J" THEN R0=101:R1=1:GOTO 90
70 IF Z$="K" THEN R0=62:R1=1:GOTO 90
80 GOTO 20
90 SOUND 0,R0:SOUND 1,R1:SOUND 7,62:SOUND 8,15
100 GOTO 20
```

●ループの時間差の解決

図7-9から，全部で20のキーを使うことがわかりますから，どのキーが押されたかを判断するために，これらのキーを配列変数に入れておきます。文字(KEN$)は，図7-9の順序でDATA文にならべておき，READ文を使って読み込みます。また，それぞれのキーが押された時にレジスタに送る値(R0, R1)も，同じ方法で読み込ませます。このとき，KEN$，R0，R1の3つの配列変数の添字は，それぞれ対応するようにしてあります。それから後の部分は前と同じ考え方です（プログラム7-19)。

〔プログラム7-19〕

```
10 DIM KEN$(20),R0(20),R1(20)
20 WIDTH80:CLS
30 RESTORE1000:FOR I=1 TO 20:READ KEN$(I):NEXT
40 RESTORE1010:FOR I=1 TO 20:READ R0(I),R1(I):NEXT
50 Z$=INKEY$(0):IF Z$="" THEN SOUND7,63:GOTO 50
60 FOR S=1 TO 20
70 IF Z$=KEN$(S) GOTO 90
80 NEXT:GOTO 50
90 SOUND0,R0(S):SOUND1,R1(S):SOUND7,62:SOUND8,15
100 GOTO 50
1000 DATA A,W,S,E,D,F,T,G,Y,H,J,I,K,O,L,P,;,":",[,]
1010 DATA 125,2,89,2,56,2,24,2,250,1,221,1,194,1,169,1
1020 DATA 145,1,123,1,101,1,81,1,62,1,44,1,28,1,12,1,253,0,238,0
,225,0,212,0
1040 DATA 100,0,94,0,89,0,84,0,79,0,75,0,71,0,67,0,63,0,59,0,56,
0,53,0
```

60行からFOR〜NEXTを使って，どのキーが押されているかを調べ，そのときのKEN$の添字の値と同じR0, R1の値をレジスタに送っています。

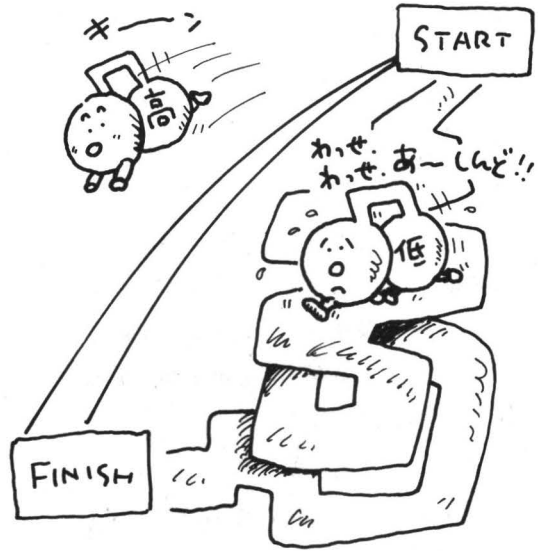
このプログラムを走らせてみると，また1つの欠点に気がつきます。キーの反応が高い音にいくほど鈍いということです。これは，FOR〜NEXT を使って，KEN$(20)を1から順に20へ向けて調べているためです。高い音ほど添字の大きい配列変数に入っていますから，たとえば，一番始めのGなどは，1回目のFOR〜NEXTループでみつかりますが，一番高いDなどは，20回ループしないとみつかりません。ループをまわる時間の差によってこのような現象が生じたわけです。

これを解決するために，INSTRという関数を使ってみます。INSTR関数の書き方は次のようになります。

INSTR(A$, B$)

この関数は，B$という文字，または文字列が，文字列A$の何文字目に含まれているかを調べる関数です。ここでは，使うキーの文字を順にならべたものをA$とし，B$に押されたキーの文字を入れ，何番目かを調べます。たとえば，B$に「H」を入れてみると，INSTR（A$, B$）の値は10になります。ここでKEN$（10）を調べてみると，やはり「H」が入っていることがわかります。このように，INSTRの値と配列の添字が一致しますから，これを利用して作り直したのが次の〔プログラム7-20〕です。

〔プログラム7-20〕

```
10 DIM KEN$(20),R0(20),R1(20)
20 WIDTH80:CLS
30 RESTORE1000:FOR I=1 TO 20:READ KEN$(I):KS$=KS$+KEN$(I):NEXT
40 RESTORE1010:FOR I=1 TO 20:READ R0(I),R1(I):NEXT
50 Z$=INKEY$(0):IF Z$="" THEN SOUND7,63:GOTO50
60 S=INSTR(KS$,Z$)
70 IF S=0 GOTO 50
90 SOUND0,R0(S):SOUND1,R1(S):SOUND7,62:SOUND8,15
100 GOTO50
1000 DATA A,W,S,E,D,F,T,G,Y,H,J,I,K,O,L,P,;,":",[,]
1010 DATA 125,2,89,2,56,2,24,2,250,1,221,1,194,1,169,1
1020 DATA 145,1,123,1,101,1,81,1,62,1,44,1,28,1,12,1,253,0,238,0
,225,0,212,0
1040 DATA 100,0,94,0,89,0,84,0,79,0,75,0,71,0,67,0,63,0,59,0,56,
0,53,0
```

30行で，文字を読み込むときに，読み込んだ文字をつないでKS$という変数を作り，60行からの部分でこのKS$に対するZ$のINSTRの値を求めています。もしKS$に含まれていない文字がZ$に入っていた場合は，INSTRの値が代入される変数Sの値は0になります。この場合は，音を出すべきキー以外のキーが押されていることになりますから，音を出す部分を通さずに50行に戻されています。

今度はかなり速いキータッチでも，きちんと音が出ます。INSTR関数を使うと，このようにいくつかの文字をまとめて判断しなければならない場合，とても効率よく処理することができます。

次のプログラムはキーボードがノーマル，キャピタルロック，カナのどのモードにあっても，「Y」と「N」に対応する処理ができるよう工夫した例で，INSTR関数とREPEAT〜UNTILを利用しています。

```
100 LOCATE12,12:CFLASH1:PRINT"G A M E  O V E R !"
110 LOCATE13,14:PRINT"Try Again ? (Y/N)"
120 CFLASH:YN$="YyンNnミ"
130 REPEAT:AN$=INKEY$:AN=INSTR(YN$,AN$):UNTIL AN<>0
140 IF AN<4 THEN CLS:BEEP:GOTO100
150 IF AN>3 THEN PRINT:PRINT"END":END
```

次は，このオルガンの音域をさらに広げて，実際のピアノやオルガンと同じ範囲の音を出せるようにします。

7-6 メモリとの直接話法

●クリック音を消そう

X１の音楽機能は，８オクターブの音域を持っています。X１のオルガンの音域も，もう少し広げてみましょう。

その前に，前のプログラムで気になっていた，キーを押すときに出るクリック音（コッコッという音）を消しておきます。それには，CLICK OFFというステートメントを使います。元にもどすには，CLICK ONとします。

まず，オルガンの鍵盤を画面に表示するようにします。サブルーチン「KEY」に処理をまとめておきます（プログラム７-21）。

〔プログラム７-21〕

クリック音の消去と鍵盤の表示の作業のため，プログラムに20行を追加します。

```
20 WIDTH80:CLS:CLICKOFF:GOSUB"KEY"
```

●音域を広げるには

さて，音域を広げるわけですが，一応，このキーボードの状態のまま，下に２オクターブ，上に２オクターブ移動した，５オクターブの音域をカバーするようにします。

まず，考えられるのは，KEN$の内容はそのままにして，R0とR1の値を20個そっくり入れかえるという方法です。それには，第一の方法として，RESTOREする行を変えて，もう一度READ～DATAで読みなおすというやりかたがあります。わかりやすい方法ですが，オクターブのアップダウンを指定したときに，RESTOREする位置の管理がむずかしそうです。

次に考えられるのは，あらかじめすべてのデータを配列に入れておき，アップダウンにした

がってこの中から適当な20組を取り出す方法です。この方法では，R0とR1をすべて読み込んでおき，添字の値をずらすことでできるはずです。ここでは，さらに一歩進めて，メモリに直接データを書き込んでおき，オクターブのアップダウンがあった場合は，読み出しをはじめるアドレスを変えることで必要な20組のデータを得られるように考えました。

メモリというのは，コンピュータ内部の記憶装置で，その中には，すべてアドレス（番地）がつけられています。X1には，0から65535までのメモリの番地があります。この中には，BASIC自身が使っているところや，プログラムの入る部分もありますから，ユーザーが自由に使えるのは限られています。ユーザーが，機械語でプログラムを作ったり，データを書き込むためには，そのためのメモリ領域を確保しておかなくてはいけません．メモリ領域を確保するためにCLEAR文を使います。CLEAR文は，オペランドに何も書かないと，変数や配列をすべて消去する働きをしますが，後にメモリのアドレスを書いておくと，そのアドレス以降は，BASICのプログラムや変数によって影響を受けなくなります（図7-10)。この部分に書き込まれたデータなどは，特に操作しないかぎり保護されています。

らんだむめも

TRON TROFF

ディズニーの「**トロン**」という映画をご存じですか。X1のコマンドにも**TRON**(トロン)というものがあります。これはTRACE ONの略で，プログラムを1行実行するごとにその行の番号を〔 〕で囲み表示するコマンドです。文字どおりプログラムの追跡をするわけです。

TRONはプログラムのデバッグ時に使うもので，エラーの出る行をさがしたり，どこでループしているか確めたりすることができます。**TRON**を解除するときはTROFF(TRACE OFF)を実行します。

しかし，プログラムの動作がおかしいからといってむやみに**TRON**を実行したところで，なかなかデバッグの効果はあがりません。画面表示の多いプログラムは，画面が見苦しくなるだけです。それより，IF文による分岐の多いプログラムで，プログラムの流れを追っていくようなときに役立ちます。

また，効果的なデバッグには，STOPをプログラムのあい間にはさんでプログラムの実行を一時停止させる方法があります。STOPを使えばCONT命令でプログラムの続きを実行できますし，マルチステートメントを使っているときも，文と文の間に入れることでどの部分でエラーが出ているか調べることができます。

このプログラムでは，&HFE00 (65024) 以降にこのエリアを指定し，この部分に，O1GからO7Dまでの音を出すために必要な67組のR0とR1を書き込んでおきます。メモリに直接データを書き込むためには，POKE文を使います。

POKE AD, D

と書くと，ADというアドレスに，Dというデータが書き込まれます。ADには，0から65535までのアドレスが使えます。CLEAR文で指定したアドレスより後にしておかないと，BASICやプログラムを壊してしまいますので注意してください。

〔図7-10〕

&H0
IOCS
モニタ
BASIC
いろいろ
~
CLEAR文によって指定されたアドレス。
この部分が機械語用エリアで、BASICの影響を受けない。
&HFF0
普通はここまでがプログラムによって使用される。
ワークエリア
&HFFFF

レジスタのデータを増し，&HFE00から交互に書き込むようにしたのが，次の〔プログラム7-22〕です。

〔プログラム7-22〕

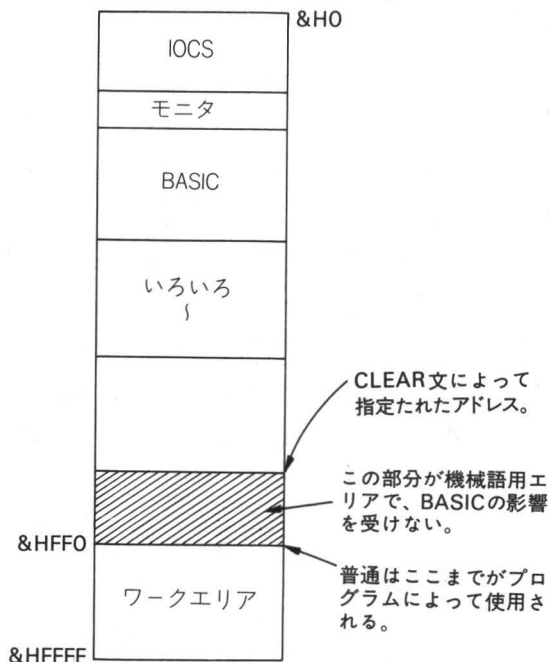
```
40 RESTORE 1010:FOR I=0 TO 67:READ R0,R1
45 POKE &HFE00+I*2,R0:POKE &HFE00+I*2+1,R1:NEXT

1000 DATA A,W,S,E,D,F,T,G,Y,H,J,I,K,O,L,P,;,":",[,]
1010 DATA 247,9,103,9,224,8,97,8,232,7,119,7,11,7,166,6,71,6,236
,5,151,5
1020 DATA 71,5,251,4,179,4,112,4,48,4,244,3,187,3,133,3,83,3
1030 DATA 35,3,246,2,203,2,163,2,125,2,89,2,56,2,24,2,250,1,221,
1,194,1,169,1
1040 DATA 145,1,123,1,101,1,81,1,62,1,44,1,28,1,12,1,253,0,238,0
,225,0,212,0
1050 DATA 200,0,189,0,179,0,168,0,159,0,150,0,142,0,134,0,126,0,
119,0,112,0,106,0
1060 DATA 100,0,94,0,89,0,84,0,79,0,75,0,71,0,67,0,63,0,59,0,56,
0,53,0
```

このプログラムを実行すると，&HFE00からのメモリの内容は，図のようになります。

〔図7-11〕

FE00	FE01	FE02	FE03	FE04	FE05	～	FE85	FE86	FE87	FE88
↓	↓	↓	↓	↓	↓		↓	↓	↓	↓
249	9	103	9	224	8	～	56	0	53	0

O3Gの音のデータは，&HFE30 (65072) に入っていますので，このアドレスから必要な20組のデータを読み出せばいいわけです。オクターブをアップダウンしたい場合には，12音分のデータ，つまり24バイトずつずらしたアドレスから読み出します。

●メモリの内容を読み出すには

メモリの内容を読み出すのにはPEEK文です。書き方は，POKE文と同じで，

PEEK AD, D

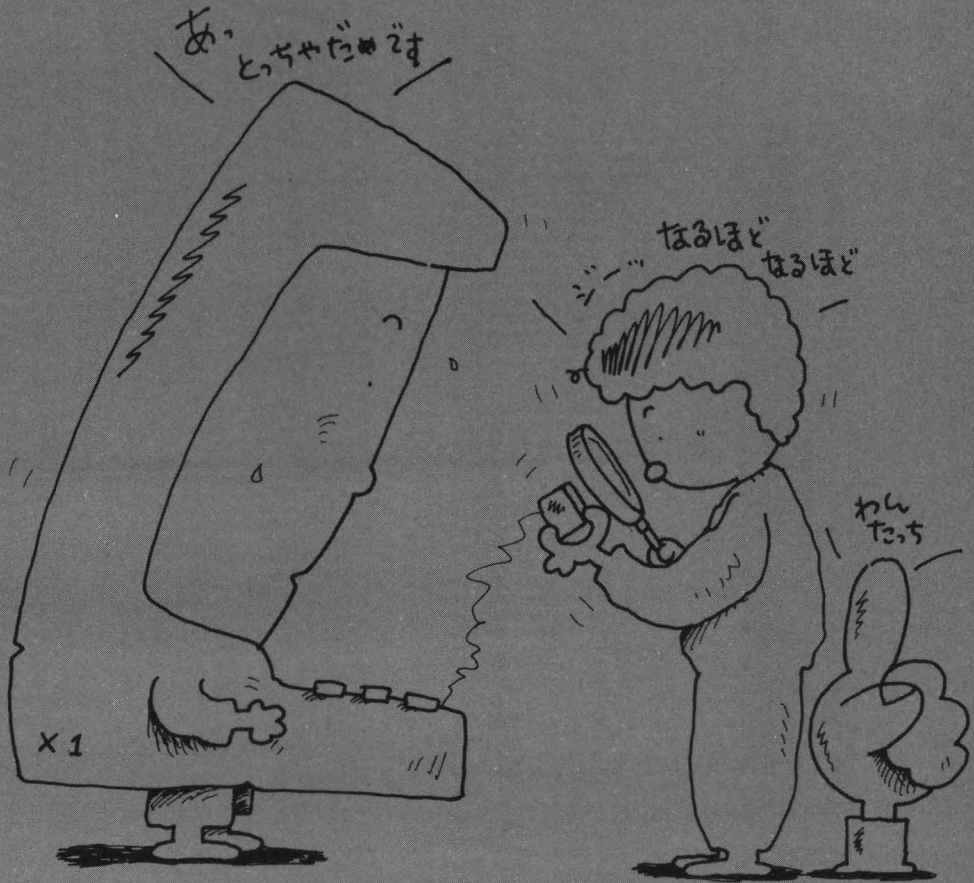
とします。オクターブのアップダウンをカーソルキーの[↑]と[↓]で行なうことにして，この指定があった時に，該当するオクターブのGの音のデータが入っているアドレスを計算して，変数SADに代入し，必要なデータを読み出すサブルーチン「READ」を呼び出すようにしたのが〔プログラム7-23〕です。サブルーチン「READ」の中では，どのオクターブが指定されているかわかるように，マークを入れるようにしてあります。また55行と57行では，最高と最低のオクターブの範囲外の指定を無視するようにしてあります。アドレスを示す「65012！」などの数値の末尾の「！」は，単精度の数を表わす記号で，打ち込まなくても自動的につけられます。

〔プログラム7-23〕

```
10 CLEAR&HFE00:DIM KEN$(20),R0(20),R1(20)
20 WIDTH80:CLS:CLICKOFF:GOSUB"KEY"
30 RESTORE1000:FOR I=1 TO 20:READ KEN$(I):KS$=KS$+KEN$(I):NEXT
40 RESTORE1010:FOR I=0 TO 67:READ R0,R1
45 POKE&HFE00+I*2,R0:POKE&HFE00+I*2+1,R1:NEXT
47 SAD=65072!:GOSUB"READ"
50 Z$=INKEY$(0):IF Z$="" THEN SOUND7,63:GOTO50
55 IF Z$=CHR$(&H1E) THEN SAD=SAD+24:IF SAD>65120! THEN SAD=65120
!:GOTO100 ELSE GOSUB "READ"
57 IF Z$=CHR$(&H1F) THEN SAD=SAD-24:IF SAD<65024! THEN SAD=65024
!:GOTO100 ELSE GOSUB "READ"
60 S=INSTR(KS$,Z$)
70 IF S=0 GOTO 50
90 SOUND0,R0(S):SOUND1,R1(S):SOUND7,62:SOUND8,15
100 GOTO50
1000 DATA A,W,S,E,D,F,T,G,Y,H,J,I,K,O,L,P,;,":",[,]
1010 DATA 247,9,103,9,224,8,97,8,232,7,119,7,11,7,166,6,71,6,236
,5,151,5
1020 DATA 71,5,251,4,179,4,112,4,48,4,244,3,187,3,133,3,83,3
1030 DATA 35,3,246,2,203,2,163,2,125,2,89,2,56,2,24,2,250,1,221,
1,194,1,169,1
1040 DATA 145,1,123,1,101,1,81,1,62,1,44,1,28,1,12,1,253,0,238,0
,225,0,212,0
1050 DATA 200,0,189,0,179,0,168,0,159,0,150,0,142,0,134,0,126,0,
119,0,112,0,106,0
1060 DATA 100,0,94,0,89,0,84,0,79,0,75,0,71,0,67,0,63,0,59,0,56,
0,53,0
1270 RETURN
2000 LABEL"READ"
2010 FOR I=0 TO 19
2020 R0(I+1)=PEEK(SAD+I*2):R1(I+1)=PEEK(SAD+I*2+1):NEXT
2025 OCT=(SAD-65024!)/24+1
2030 CSIZE2:FOR I=1 TO 5:LOCATE78,I:PRINT#0,"o":NEXT
2040 LOCATE78,6-OCT:PRINT#0,"●"
2050 RETURN
10000 LABEL"KEY"
10010 LOCATE0,15
10020 PRINT" ■■■ ■■■■■ ■■■■■  |  ■■■■■ ■■■■■  |  ■■■■■ ■■■■
■ ■■■■■  |  ■■■■■ ■■■
10030 PRINT" ■■■ ■■■■■ ■■■■■  |  ■■■■■ ■■■■■  |  ■■■■■ ■■■■
■ ■■■■■  |  ■■■■■ ■■■
10040 PRINT" ■■■ ■ W ■ ■ E ■  |  ■ T ■ ■ Y ■  |  ■ I ■ ■ O
■ ■ P ■  |  ■ [ ■ ■■■
```

```
10050 PRINT"  ███ █████ █████    |    █████ █████    |    █████ █████
█ █████    |    █████ ███
10060 PRINT"  |     |     |     |     |     |     |     |     |
    |     |     |     |
10070 PRINT"  |     |     |     |     |     |     |     |     |
    |     |     |     |
10080 PRINT"  |  A  |  S  |  D  |  F  |  G  |  H  |  J  |  K  |
  L  |  ;  |  :  |  ]  |
10090 PRINT"  |     |     |     |     |     |     |     |     |
    |     |     |     |
10100 PRINT"  ┴─────┴─────┴─────┴─────┴─────┴─────┴─────┴─────┴──
─────┴─────┴─────┴─────┴
10110 RETURN
```

以上のプログラムでO1GからO7Gまでの音階を演奏することができます。普通のピアノが77鍵ですから，かなりのものになります。あとは，弾いた音楽を記憶しておけるようにするとか，自動的に楽譜を書くなどの拡張を加えてみてください。

X1機能のいろいろ

8-1　ワンタッチキーを使う

●ワンタッチキーの使い方

Ｘ１のキーボードには，５個のファンクションキーが用意されています。SHIFTキーとあわせて使うことで，５個のキーが10個分の働きをすることはみなさんご存じのとおりです。これらのキーには，ひんぱんに使うコマンドなどが定義されているということも，もうおなじみになっているでしょう。

ファンクションキーに定義されているコマンドなどの内容を，KLISTまたはKEYLISTという命令で画面に表示することができます。これらのコマンドや関数以外にしばしば使うものがある場合に，ファンクションキーの内容を自由に定義し直して使うことができます。ファンクションキーの定義を行なうには，

(1)KEYn，文字列

(2)DEF KEY 文字列

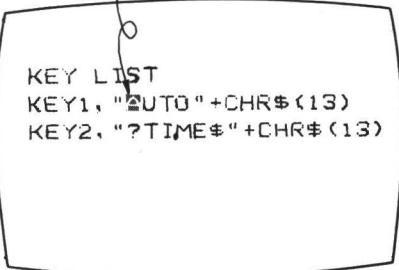
という２つの命令がありますが，どちらもまったく同じ働きをします。(1)の書き方で［F 1］キーにFILES ↵を定義するには，

KEY1, "FILES" ＋CHR$(13)

とします。CHR$(13)は↵キーを表わすコントロールコードです。この形式は，KEY LISTによって画面に表示されるものとまったく同じ書式になっていますから，キーリストを表示しておいて，その表示を表きかえることによって定義を行なうことができます（画面８-１）。

ファンクションキーには，最大15文字までの命令を定義することができます。15文字を越えて定義しようとしても16文字以降は無視されてしまいます。

〔画面８-１〕

KLISTを表示して，KEY4のところを見ると，後に，CHR$(26, 13)とあります。このCHR$(26)は［CTRL］＋［Z］キーを押したのと同じ働き，つまりカーソルから後をすべて消去するコントロールコードです。これらのコントロールコードも１文字として数えられます。たとえば，KEY 5，"RUN" ＋CHR$(13)は，「RUN」が３文字，リターンキーを表わすコントロールコード，CHR$(13)が１文字の合計４文字を定義していることになります。２つ以上の命令をマルチステートメントで定義する場合など，文字数が足りなくなることがありますが，このような場合は省略形を使うことで文字列を短くすることができます。たとえば，

WIDTH80：LOAD：RUN↵

と定義したい場合，このまま打っていくと15文字ですからRUNのUまでしか入りませんが，

KEY1, "WI. 80：LO.：R. "＋CHR$(13)

とするとすべての命令を１つのキーに定義することができます。また，

PRINT "A"⏎

と定義するときに

KEY1, "PRINT"A""

のように，ダブルクォーテーションマークを重ねて使うとエラーになります。このような場合は，「"」のコードであるCHR$(34)を使って，

KEY1, "PRINT"＋CHR$(34)＋"A"＋CHR$(34, 13)

とすることで定義できます。このように定義したものは，キーリストを取った場合にも定義したときと同じように，CHR$( )の形で表示されますが，実際に F1 キーを押すと，

PRINT"A"

と表示され，この命令が実行されます。

●ファンクションキーもう１つの用法

X１のファンクションキーは，以上のように文字列を定義する本来の機能の他に，ON KEY GOSUB文を使うことで，１から10までの分岐先の指定に使うことができます。ON KEY GOSUB文は，１から10までのファンクションキーが押された時点でそのキー番号に従ってサブルーチンへの分岐を行なうものです。たとえば，

ON KEY GOSUB 100, 200, 300

となっている場合，ファンクションキーの１番を押すと100行から，２番を押すと200行からのサブルーチンを呼び出します。ON KEY GOSUB文による分岐先は，行番号を使い，ラベルを使うことができません。ON KEY GOSUB文の大きな特徴は，このステートメントにプログラムの流れが来たときに実行されるのではなく，プログラム中で一度指定した後は，プログラムが他の作業をしている途中でもファンクションキーが押されるとサブルーチンへの分岐が実行されるという点です。つまり，ON KEY GOSUB文で分岐先を決めておくと，ファンクションキーを押すことによって「割り込み処理」が行なわれます。〔プログラム８－１〕がON KEY GOSUBを利用したプログラムです。

〔プログラム８－１〕

```
10  WIDTH 40:SCREEN 0,0:CLS 4
20 ON KEY GOSUB 50,110,170
30 FOR L=0 TO 255:COLOR L MOD 7+1:PRINT#0 CHR$(L);:PAUSE 1:NEXT
40 FOR L=1 TO 255:PRINT "※";:NEXT:GOTO 30
50 LABEL "ｴﾝ"
60 CLS 4:FOR I=1 TO 6
70 GOSUB "PARA"
80 CIRCLE(X,Y),R,C
90 NEXT
100 RETURN
110 LABEL "ｼｶｸ"
120 CLS 4:FOR J=1 TO 6
130 GOSUB "PARA"
140 POLY(X,Y),R,C,90,0,360
150 NEXT
160 RETURN
170 LABEL "ﾛｯｶｸ"
180 CLS 4:FOR K=1 TO 6
```

```
190 GOSUB "PARA"
200 POLY(X,Y),R,C,60,0,360
210 NEXT
220 RETURN
230 LABEL "PARA"
240 X=INT(RND*200)+60:Y=INT(RND*100)+50
250 R=INT(RND*100):C=INT(RND*7)+1
260 RETURN
```

このプログラムを走らせると，すべてのキャラクタの色を変えながら描く処理と，「＊」を255個描く処理を繰り返していますが，ファンクションキーの1から3を押すと，画面をクリアしてから1なら円を，2なら四角形を，3なら六角形をそれぞれ6個ずつ描きます。ON KEY GOSUB文で割り込みが行なわれても，元のルーチンの変数は保護されています。ですからキャラクタを描いている途中にファンクションキーが押されて割り込みが行われた後も，元のルーチンへ戻っていくとその次のキャラクタから再び描き始めます。また，サブルーチンを実行して図形を描いているときに，ファンクションキーを押すとそこからさらに分岐します。この場合も最初に実行されたサブルーチンに戻って来たときには，残っている数だけ図形を描いてから元のルーチンに戻ります。

このような動作を確実に行なわせるためには，メインルーチンとサブルーチンの変数が重ならないようにしておくことが大切です。

〔プログラム8-1〕の場合，[SHIFT]＋[BREAK]キーで実行を止めると，ON KEY GOSUBで使われた[F1]から[F3]のファンクションキーを押しても本来の働きを行ないません。これを元に戻すには，[CTRL]＋[D]を押すか，KEYOFFという命令を実行します。KEYOFFは，KEYON，KEYSTOPとともに，ON KEY GOSUB文をコントロールするステートメントです。ON KEY GOSUB文が実行されたあとでも，KEYn OFF（nはファンクションキーの番号）が実行されると，nで指定したキーについては本来の機能を持つことになります。nを省略するとすべてのファンクションキーが本来の機能に戻ります。

●KEYnSTOPで保留する

KEYnSTOPは，一時的にON KEY GOSUBの働きを保留するもので，KEYnONが実行されると解除され，ON KEY GOSUBによる分岐が行なわれるようになります。〔プログラム8-1〕に，次の3行を追加してください。

```
25 KEY 2 STOP:KEY 3 STOP
95 KEY 2 ON
155 KEY 3 ON
```

こうして実行すると，初めは円を描くサブルーチンにしか分岐できませんが，一度ここへ分岐すると95行で[F2]キーがONとなるため，[F2]キーの分岐が機能します。さらに，ここへ分岐すると155行で[F3]キーもONになるため，3つのファンクションキーすべてが機能します。KEYON，KEYOFF，KEYSTOPは，キー番号を指定するとそのキーだけに働き，省略すると10個のファンクションキーすべてに働きます。

ゲームの途中などでキーの説明を見たい場合など，ON KEY GOSUBを利用すると便利です。この場合，分岐が行なわれてはいけない部分にはKEYn STOPを入れておけば，不都合なことは起きないで済むようになります。

8-2 先取りキーの働き

●先取りキー（KEY 0）

X 1には先行キー入力バッファという機能があります。

```
KEY0,"FOR I=0 TO 255:PRINT#0 CHR$(I);:NEXT"+CHR$(13)
```

と打ち込んでみてください。リターンキーを押すとダブルクォテーションで囲んだ部分が画面に表示されると同時に，そのプログラムが実行されて255個のキャラクタが表示されます。これが先行キー入力バッファの働きです。先行キー入力バッファに文字を書き込んでおくと，次に入力待ちの状態となったときに（ここでは画面に「OK」と表示されたときに），ダブルクォテーションで囲んだ文字列をキーボードから打ち込んだのと同じ働きをすることになります。

先行キー入力バッファに文字を書き込むには，

KEY0, "文字列"

とします。ファンクションキーと同じように，CHR$(13)などのコントロールコードも書き込むことができます。この先行キー入力バッファの文字列は，プログラムが入力待ちの状態，つまり，INPUT文で止まっている時や，INKEY$関数でキーが押されるのを待っている状態になるまで保持されており，このような状態になった時に画面上に出力されます。したがって，これらの命令の前に，KEY 0の命令を書き込んでおくと，入力があったのと同じ状態になります。

次のプログラムを走らせてみてください（プログラム8-2）。

〔プログラム8-2〕

```
10 WIDTH 40:SCREEN 0,0:CLS 4
20 LOCATE 0,0:INPUT "AUTO=1  MANUAL=2 ";AM
30 IF AM<>1 AND AM<>2 GOTO 20
40 IF AM=1 THEN KEY0,"1"+CHR$(13)
50 LOCATE 0,2:INPUT "ｴﾝ=1  ｼｶｸ=2  ﾛｯｶｸ=3 ";N
60 ON N GOSUB "ｴﾝ","ｼｶｸ","ﾛｯｶｸ"
70 GOTO 50
80 LABEL "ｴﾝ"
90 CLS 4:FOR I=1 TO 6
100 GOSUB "PARA"
110 CIRCLE(X,Y),R,C
120 NEXT
130 IF AM=1 THEN KEY0,"2"+CHR$(13)
140 RETURN
150 LABEL "ｼｶｸ"
160 CLS 4:FOR I=1 TO 6
170 GOSUB "PARA"
180 POLY(X,Y),R,C,90,0,360
190 NEXT
200 IF AM=1 THEN KEY0,"3"+CHR$(13)
210 RETURN
220 LABEL "ﾛｯｶｸ"
230 CLS 4:FOR I=1 TO 6
240 GOSUB "PARA"
250 POLY(X,Y),R,C,60,0,360
260 NEXT
270 IF AM=1 THEN KEY0,"1"+CHR$(13)
280 RETURN
290 LABEL "PARA"
```

```
300 X=INT(RND*200)+60:Y=INT(RND*100)+50
310 R=INT(RND*100):C=INT(RND*7)+1
320 RETURN
```

初めにAUTOかMANUALかを聞いてきますが，MANUALにすると，円，四角，六角どれにするかを選ぶようになります。AUTOにしておくと，まず40行でKEY 0 に〝1″＋CHR$(13)が入りますから，50行のINPUT文で1↵とキーボードから入力したのとまったく同じ働きをしますから，60行で円を描くサブルーチンを呼び出します。このサブルーチンでは，RETURN文の前の130行で，KEY 0 に〝2″＋CHR$(13)が書き込まれますから，50行に戻った時に2↵と入力したことになり，四角を描くサブルーチンを呼び出します。ここでも同じようにして先行キー入力バッファからの入力が行なわれるようになっていますから，結果的には，円→四角→六角→円→四角……と繰り返していることになります。

このように，キー入力バッファを使うとキー入力を行なわなくても，キー入力があったのと同じ状態を作ることができますから，ゲームのデモ画面をわざわざ別に作らなくとも，メインルーチンを自動進行させることによってデモを行なうことができます。

●プログラムの自己増殖

先行キー入力バッファに書き込めるのは，最大64文字ですから，うまく使ってやるとおもしろいことができます。特に，X1ではプログラムの修正を行なっても変数が保護されていますから便利です（普通BASICではプログラムの修正や行の追加を行なうと変数はクリアされます)。というのは，BASICでは，先頭に数字のついた文字列はすべてプログラム上の「行」として扱われますから，先行キー入力バッファを利用して，プログラムを自己増殖させることができるのです。たとえば，

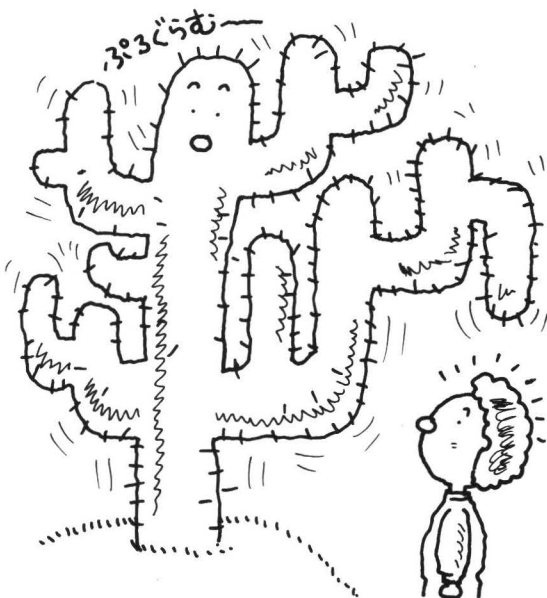
KEY0, "1000 PRNIT A"＋CHR$(13)↵

と打ってみてください。LISTを取ると今までになかった1000行目が作られています。このことをうまく利用すると，入力したものをDATA文として次々に追加していくことができます。これが〔プログラム8-3〕です。ここでは，名前と電話番号を入力すると，

行番号　DATA　名前，電話番号

という形でプログラムに次々とつけ加えられていきます。

〔プログラム8-3〕

```
10 WIDTH40:SCREEN0,0:CLS
20 GYO=1000
30 LOCATE0,0:INPUT"NAME       ",NAM$
40 IF NAM$="END" THEN SCREEN0,1:KEY0,STR$(GYO)+" DATA "+"END,END
"+CHR$(13)+"20 GYO="+STR$(GYO)+CHR$(13):SCREEN0,0:END
50 LOCATE0,4:INPUT"TEL.No.  ",TEL$
```

```
60 SCREEN0,1
70 KEY0,STR$(GYO)+" DATA "+NAM$+","+TEL$+CHR$(13)+"GOTO90"+CHR$(
13)
80 END
90 GYO=GYO+10
100 SCREEN0,0:CLS:GOTO30
```

KEY 0 で先行キー入力バッファに

　　行番号　DATA　名前，電話番号⏎GOTO90⏎

という内容が書き込まれます。この内容は，80行目のEND文によって画面に表示されます。すると，同じ内容をキーボードから打ち込まれたときの働きをします。

行番号は，変数GYOによって管理されていますから，次々と新しい行が作られていくことになります。このようにしてDATA文をつけ加えたプログラムをセーブしておけば，次回からはREAD文でこのデータを読み込むことによって，データを増やしていくことが可能です。ここで注意しておきたいのは，70行にEND文を置いていることです。これがないと，先行キー入力バッファの内容が出力されないため，何の意味もなくなってしまいます。また，先行キー入力バッファによってプログラムが書き込まれる様子は見られませんが，これは60行のSCREEN文によって，表示されていない方の画面上で作業しているためです。また，入力を終るには名前に「END」と入れますが，このとき最後のデータとして

　　行番号　DATA END, END

が書き込まれ，さらに20行の変数GYOの値も書き直していますから，もう一度RUNをしてもそれまでにあったものの後に新しい行がつけ加えられていくわけです。

8-3 いろいろな関数

これまでにもいろいろな関数を使ってきましたが，X 1のBASICには，それらの他にも科学技術計算などに活用できる，数値関数が用意されています。数値関数をまとめて見るとともに，グラフィック機能を利用して，これらの関数のグラフ化なども試してみましょう。

一般的に，関数は次のような書式で使われます。

Y＝FN(X)

FNの部分が関数名で，SIN, COSなどを書きます。Xは求める対象となる数値で，「引数」と呼びます。この引数は，それぞれの関数によって取り扱える範囲がありますから注意してください。

●三角関数

三角関数には，SIN，COS，TANがあります。引数にはラジアン単位を使います。角度(DEG)をラジアンに変換するには，RAD関数を使います。図8－1は，0から720度のSIN, COS, TANの値をグラフ化したもので，三角関数の引数にはＩの値をRAD関数によってラジアンに変換したものを使用しています。関数の値によって，グラフィック座標に対する補正を行なってる点に注意してください。三角関数の引数には，負の値や，数式を使うこともできます。

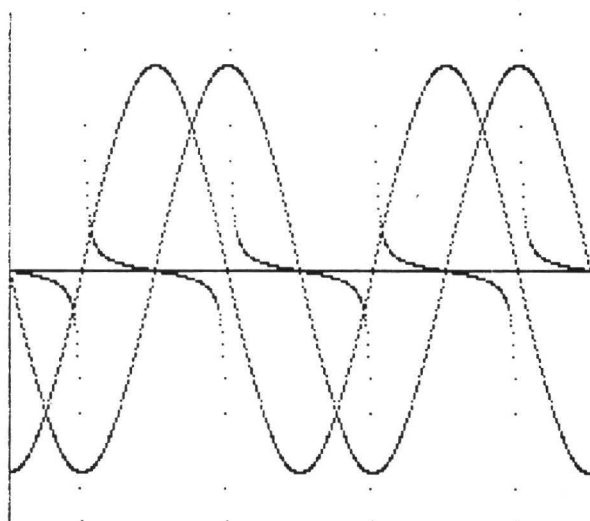
〔図8－1〕

140行で，RAD関数によって，Ｉの値をラジアンに変換し，変数THETAに入れています(プログラム8－4)。「360°＝2πラジアン」ですから，140行を省略し，130行を

```
FOR THETA=0 TO PAI(4) STEP PAI(1)/64
```

とすることもできます。

〔プログラム8－4〕

```
100 WIDTH 40:INIT:SCREEN 0,0:CLS 4
110 LINE(0,100)-(240,100),PSET,7
120 LINE(0,0)-(0,199),PSET,7
130 FOR I=0 TO 720 STEP 1
140 THETA=RAD(I)
150 X=I/3
160 YS=SIN(THETA)*80+100
170 YC=COS(THETA)*80+100
180 YT=TAN(THETA)*3+100
```

ます。

```
190 IF YT>198 THEN YT=199 ELSE IF YT<1 THEN YT=0
200 PSET(X,YS,2)
210 PSET(X,YC,4)
220 PSET(X,YT,6)
230 NEXT
240 HCOPY 0
```

PAIは，円周率のｎ倍の値を求める関数です。PAI（１）＝3.1415927です。このPAI関数はグラフィック文字の「π」で代用することができます。綿密な計算を行ないたいときは，π#として，倍精度にすると，

π#＝3.141592653589798

として使うことができます。この場合，他の変数も倍精度にしておく必要があります。X１の数値変数は通常有効桁数が８桁ですが，変数名の後に#をつけると，有効桁数が16桁の倍精度になります。プログラムの初めに，DEFDBLという命令を使えば，指定のアルファベットから始まる変数すべてが倍精度に宣言されます。

DEFDBL A-Z（すべての数値変数を倍精度で使う）
DEFDBL X, Y（XとYで始まる変数名のうち数値変数のみ倍精度で使われる）

この場合，倍精度に指定した変数名には「#」をつける必要はありません。また，指数表示の時には「E」ではなく「D」が表示されます。

この逆に，有効桁数８桁に指定するには，変数名の後に「！」をつけるか，DEFSNG文で宣言しておきます。

これらの指定の他に，似たもので，DEFINTとDEFSTRの命令があります。

ゲームを作るときなど，小数点以下の細かい数値を無視してもよい場合があります。このようなときは，変数を整数で使うように指定しておくと，計算の速度を上げることができます。変数名の後に％をつけると整数として扱われます。この宣言を行なうのがDEFINT文です。

また，プログラム中で，文字列変数をたくさん使う場合，DEFSTR宣言を行なうと，その文字で始まる変数には今までのように「$」をつける必要がなくなります。

これらの変数の型の宣言は，あらかじめどの変数にどのようなデータが入るかを整理しておいて使うと効率のよいプログラム作りができます。

三角関数に関連して，ATN（X）があります。これは，数値Xに対するアークタンジェントを求める関数です。アークサイン，アークコサインは関数として用意されていませんが，他の関数と組み合わせて，ユーザー定義関数として使うことができます。

アークサイン………DEFFNA(X)＝ATN(X／SQR(1－X∧2))

アークコサイン……DEFFNB(X)＝－ATN(X／SQR(1－X∧2))＋PAI(1)／2

これらを誘導関数と呼びます。この他にも，誘導関数によって求められる関数がいくつかありますが，これらの使い方を《8－3》，《8－4》にまとめてありますから利用してください。

SQR(X)は，平方根を求める関数です。Xの値は負になると，当然エラーになります。平方根（2乗根），多重根以外は，次のようにすると求めることができます。

$\sqrt[Y]{X}$＝X∧(1／Y)

例：$\sqrt[3]{X}$＝X∧(1／3)

●指数関数

EXP(X)と書かれます。指数関数は，自然対数の底e（e≒2.7182818）のX乗を求める関数です。X１で扱える最大の数は1.7×10^{38}までですから，Xの値は－88から88まででこの範囲を越えるとエラー（オーバーフロー）になります。

●対数

LOG(X)と書きますが，求められるのは自然対数（数学ではln）になります。10を底とする常用対数を求めるには

LOG(X)／LOG(10)

とします。

●階乗など

FAC(X)と書き，X！の値，つまり，１からXの値すべてをかけたものを求めます。Xの値は，０から33までです。

SUM(X)

１からXまでの数値の和を求めます。Xの値は，０から1×10^{19}までです。

●その他の数値関数

〈絶対値〉

ABS(X)とし，Xの絶対値を求めます。

〈符号〉

SGN(X)とし，Xの値が負，０，正であると，SGN(X)の値は，次のようになります。

X＞０のとき　SGN(X)＝１

X＝０のとき　SGN(X)＝０

X＜０のとき　SGN(X)＝－１

〈整数化〉

INT(X)……数値Xを超えない最大の整数を求めます。

FIX(X)……数値の整数部を取り出します。つまり，小数点以下を切り捨てた値を求めることができます。

INT(X)の場合「Xを超えない」ということですから，Xの値が負のときに，INT(X)とFIX(X)の値は違ってきます。

INT（－3.14）＝－４

FIX（－3.14）＝－３

となります。

数値データを任意の桁で四捨五入するには，小数点以下N＋１桁目を四捨五入して，小数点

以下N桁まで求めるとすると，

Y＝INT(X×10∧N＋0.5)／10∧N

とします。0.5を0.9にすると切り上げになります。

8-4 文字関数

ここでは，前に出てきたものも含めて，文字処理の関数をもう一度まとめておきます。

それぞれ，まったく逆の働きをするものや，関連のあるものと対比しながら説明していきます。

●ASC⟷CHR$

ASC関数は，文字の持っているキャラクタコードを数値に変換します。書式は，

```
A=ASC(A$)
```

のようにします。

X＝ASC("A")とすると，Aのキャラクタコードである65がXの値になります。

```
A$="ABC" : Y=ASC(A$)
```

のように，文字列の場合は，左端にあるAのキャラクタコード65がYの値になります。この逆に，数値をキャラクタコードとする文字を求めるのがCHR$関数です。通常

```
A$=CHR$(65)
```

のように使いますが，カッコの中にカンマで区切って数値を並べると，その数値に対応する文字列が求められます。たとえば，

```
PRINT  CHR$(83, 72, 65, 82, 80, 32, 88, 49)
```

とすると，「SHARP　X1」という文字列が書かれます。32は，スペース（空白）のキャラクタコードです。

●VAL⟷STR$

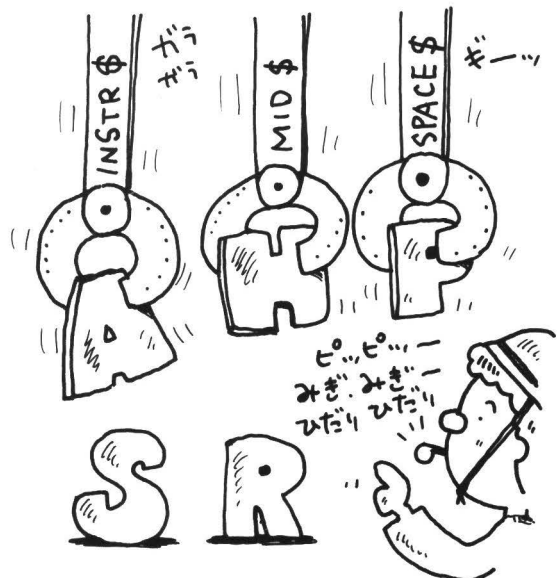
VAL関数は数字でできた文字列を数値に変換します。「＋」と「－」，そして，16進数や２進数を示すときに使う「&」以外の文字が先頭にあった場合，VALの値は０になります。また，文字列の途中に数字以外の文字が入った場合，それ以後は無視します。たとえば，

```
A=VAL("325K63")
```

の場合，変数Aの値は，325になります。

VAL関数は，16進数などを10進数に変換するときに便利に使えます。

```
PRINT VAL("&H" + "FC20")
```

16進数のFC20が10進数に変換されます。このような場合，&Hをつけて16進数として扱われますから，数字以外のAからFも有効になります。この例の場合，値として－992

が表示されます。これは，&H8000以上の16進数は，10進数に直した場合，65536の補数として表わされるためです。ですから，実際に10進数に換算すると64544となります。メモリのアドレス計算の場合，X１のフリーエリアはすべてこの補数表現の範囲になりますから，

65536－VAL(“&H”＋AD$)

のようにしてやる必要があります。ただし，POKE文やPEEK文などでアドレスを指定するときには，直接この負の値を入れても問題ありません。

VAL関数とは逆に，数値を文字列に変換するのが，STR$関数です。数値の先頭には必ず符号桁があります。正の数値の場合，先頭に「＋」の符号が表示されない分のスペースが入ります。したがって，実際に得られる文字列の長さは，数字の桁数＋１の長さになります。

●HEX$, OCT$, BIN$

これらはそれぞれ，数値を16進数，８進数，２進数の文字列に変換するものです。定数の場合，先頭に16進数は&H，８進数は&O，２進数は&Bをつけます。

&Bは，２進数の０と１の組み合わせの文字列ですから，グラフィックのプライオリティや，SOUND文のレジスター７の設定に使うとわかりやすくなります。たとえば，グラフィックスのプライオリティを，青と黄色のグラフィック優先にしたい場合，２進数の各桁が，パレットコードに対応しますから，

PRW　&Ｂ０１０００１０

のように書くことができます。SOUND文のレジスタ７の場合，音を出すところを０にしますから，チャンネル１から音を，チャンネル２からノイズを出す場合，

SOUND　7, &B010001

としてやればよいわけです。

●LEFT$, RIGHT$, MID$, INSTR

これらについては，「テレビ24時間予約」のプログラムの中で，詳しく説明しました。この中で，MID$関数には，少し変わった使い方があります。MID$関数は，一般的に

MID$ (A$, 3, 2)

のように使いますが，文字数を指すパラメータを省略すると，３文字目から後ろ全部を取り出すことができます。RIGHT$関数では，右から何文字かを指定して取り出しますが，この方法を使うと，左から何文字目以降を文字列の最後までを無条件に取り出すことができます。〔プログラム８-５〕はこのことを利用して，計算式を文字列に入力し，この式の値を計算するものです。

170行で，符号（演算子）以降の文字を取り出すときにこの方法を使っています。

〔プログラム８-５〕

```
100 CA$(1)="+":CA$(2)="-":CA$(3)="*":CA$(4)="/"
110 INPUT "ｹｲｻﾝｼｷ ",C$
120 FOR I=1 TO 4
130 FOR J=1 TO LEN(C$)
140 IF MID$(C$,J,1)=CA$(I) THEN GOTO 160
150 NEXT:NEXT
```

```
160 K=I:Z=INSTR(C$,CA$(K))
170 X$=LEFT$(C$,Z-1):Y$=MID$(C$,Z+1)
180 ON K GOSUB 210,220,230,240
190 PRINT C$;" ノ コタエハ ";ANS
200 END
210 ANS=VAL(X$)+VAL(Y$):RETURN
220 ANS=VAL(X$)-VAL(Y$):RETURN
230 ANS=VAL(X$)*VAL(Y$):RETURN
240 ANS=VAL(X$)/VAL(Y$):RETURN
```

160行のINSTR関数は，文字列の中に，ある文字列が何文字に入っているかを調べる関数です。

一般的には

A=INSTR（n, A$, B$）

と書きます。nは，A$の左から何文字目からB$を探しはじめるかを決める数値で，省略すると最初の文字から順に探していくことになります。A$の中にB$が含まれていない場合，値は0になります。

もう1つ，MID$には関数としてではなく，ステートメントとしてのMID$もあります。たとえば，

MID$（A$, 3, 1）="A"

とすると，A$の3文字目がAに置きかえられます（図8－2)。一般形では，

MID$（A$, m, n)=B$

と書きます。mは，置きかえをはじめる位置，nは，B$の左から何文字で置きかえるかを指定するものです。mの値がA$の長さを越えている場合，何も変化しません。また，nの値がB$の長さより大きくなった場合，エラーになるわけではなく，B$の長さで処理されます。

〔図8－2〕

●STRING$, SPACE$

SPACE$は，希望の文字数のスペースをつくります。

SPACE$(n)

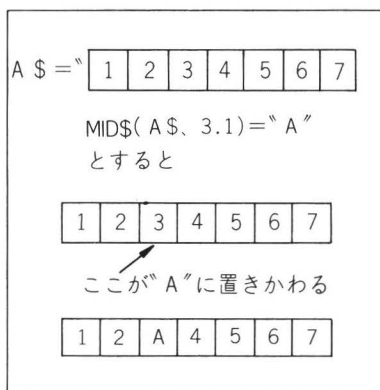
と書き，nはスペースの数を指定する数値で0から255まで使うことができます。スペース以外の文字や，文字列を希望する数だけ連続させたいときには，STRING$関数を使います。

STRING$関数には

STRING$(n, A$)	①
STRING$(n, A)	②

の2とおりの使い方があります。①の場合，A$に入っている文字列をn個つないだものになります。②のように，数値（ここでは数値変数A）を書くと，その数値のアスキーコードを持つ文字がn個つなげられたものとなります。つまり，

SPACE$(n)=STRING$(n, 32)=STRING$(n, "　")

ということになります。

●LEN

LENは文字列の長さ(文字数)を求める関数です。〔プログラム 8 - 5〕の130行では，FOR〜TO〜NEXTの終了値を決めるのに，LENを使っています。

●HEXCHR$

CHR$と似ていますが，こちらはカッコの中に16進数 2 桁のならびの文字列を入れます。すると，この文字列の左から 2 文字ずつ区切って，その16進数の示すキャラクタコードに対応する文字に変換します。したがって，

HEXCHR$("414243")＝CHR$ (&H41, &H42, &H43)

ということになります。HEXCHR$関数は，数値の両端をダブルクォーテーションで囲い，文字列として使うことに注意してください。この文字列には，区切りとして，スペースを置くことができ，このスペースは無視されます。

8-5　データ保存とファイル処理

●2つのファイル機能

今まで，いろいろなデータをパソコンに入力し，それを表やグラフに表現する方法を学んできました。これらの処理は，データ入力──→処理──→表示（出力）という一連の作業でした。ところが，実際のデータの中には，この一連の作業が終ったあとも保存しておき，あとからもう一度別の処理を加えたり，処理した結果によって再検討しなければならないことがあります。このような場合，データをファイルとして保存しておくことが必要になります（保存)。

またパソコン本体のメモリに入り切らない多量のデータを扱うときなど，そのデータをファイルに書き込んでおき，少しずつ読み出して処理するような方法も考えなくてはいけません(補助記憶)。

もう1つは，各地に散らばる支店から集めたデータを一括して本店で処理するといった，別の場所から得られたデータをまとめて全体の様子を調べるような場合も考えられます（データの集積)。

このように，ファイルを有効に使うと，さまざまなメリットがあります。

ここでは，X1本体でできるカセットテープによるファイル作成について見ていきます。ファイルには，「シーケンシャルファイル」と「ランダムファイル」があります。

シーケンシャルファイルでは，データはファイルの先頭から順番に記録（格納）されていて，読み出すときにはたとえ不必要なデータがあったとしても，順番に読み出していかなくてはなりません。

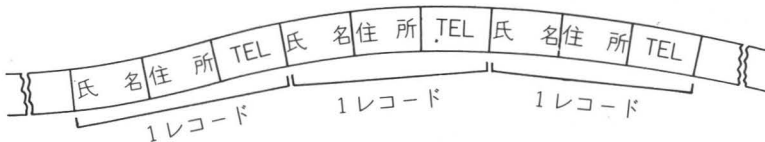
ランダムファイルでは，希望のデータをファイルの中から自由に取り出すことができます。これはちょうどLPレコードとカセットテープの関係に似ています。レコードでは，聞きたい曲の先頭に針をおろすだけで選曲できますが，カセットテープではその曲の先頭まで，順にテープを送っていかなくてはなりません。

ランダムファイルでは，希望のデータをファイルの中から自由に取り出すことができます。これはちょうどLPレコードとカセットテープの関係に似ています。レコードでは，聞きたい曲の先頭に針をおろすだけで選曲できますが，カセットテープではその曲の先頭まで，順にテープを送っていかなくてはなりません。

X1ではBASICを与える方法に，カセット，ディスク，ROMの3とおりがあります。カセット版BASIC(CZ8CB01)では，ランダムファイルはグラフィックメモリと外部メモリに，シーケンシャルファイルはカセットテープ，グラフィックメモリ，外部メモリに作成することができ

●アドレスブックを作ろう

ここでは，ファイルの機能を使いながら「アドレスブック」を作ってみましょう。まず氏名，住所，電話番号を入力しておき，氏名を入力することで住所や電話番号を調べられるようにします。１人分のデータは氏名，住所，電話番号となります。このようなデータのまとまりを１レコードと呼びます。テープ上へ１レコードのデータの書き込まれる様子を図に示すと図８-３のようになります。

〔図８-３〕

実際にこのようなファイルを作るステートメントを見ていきましょう。

まず，ファイルを作成するためには，データをカセットテープに送り出さなければなりません。このためには，カセットテープのファイルを使用できるように用意をしてファイルの書き込みを行なうのか，読み出しを行なうのかも指定します。このためのステートメントがOPENです。

まず書き込みを行なうには，

```
OPEN "O", #n, "CAS:name"
```

とします。nはファイル番号で，X１ではファイルの取り扱いはすべてこのファイル番号で指定します。一度に使用できるファイル番号は，０から15までで，いくつまでのファイルを使うかは，MAXFILESという命令によって指定しておきます。この指定をしないときは，nの値は自動的に１になります。

“O”は，パソコンからデータをOutputする，つまり「テープへの書き込みのためのファイルOPEN」であることを示します。ここでは，ファイルディスクリプタはカセットを使いますから，「CAS：」としています。もちろん，この部分を他の外部記憶装置にすることもできます。たとえば，「MEM：」とすれば，グラフィックメモリを外部記憶装置として使うことになるわけです。

●テープへのデータ書き込み

テープへのデータの書き込みは，PRINT＃，WRITE＃の２とおりの命令があります。PRINT＃では，複数のデータを書き込む場合，区切りとして「,」や「；」を使うことができます。しかし，データが文字変数の場合，これではかえって都合の悪いことになってしまいます。たとえば，PRINT＃1, A$; B$; C$としたとすると，セパレータのセミコロンの働きにより，テープ上に３つの文字変数が連続して書き込まれてしまいます。ですから，読み出したときには３つの別々のデータではなく，ひとつづきのものとして読み出されてくることになります。

一方WRITE＃を使うと，それぞれ別のデータとして記録されます。WRITE＃で，データはセミコロンやカンマをつけたまま書き込まれ，読み出すときにもこの形式で読み出されますから，並べて書き込んでもそれぞれが別のデータとして扱うことになります。ただし，数値データの場合はPRINT＃でも，別のデータとして読み出しが行なわれます。

ファイルのデータの書き込みが終了したら，OPENの反対に，ファイルの使用を終わるという指示をしなくてはいけません。これがCLOSE文です。後にファイル番号を付けると，そのファイルだけが閉じられ，省略すると使用中のすべてのファイルが同時に閉じられます。

●パソコンアドレスブック

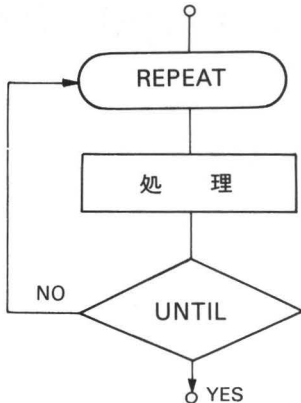
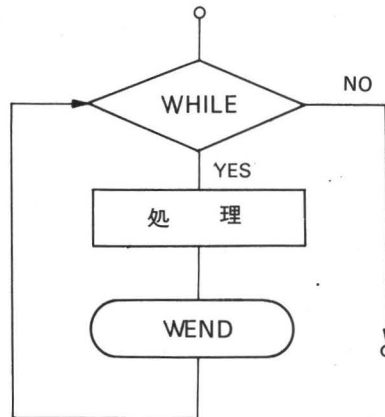
「パソコンアドレスブック」のプログラムリストを見てみましょう。

〔プログラム8-6〕

```
10 OPTIONBASE1:DIM NAM$(255),ADRS$(255),TEL$(255)
20 WIDTH40:SCREEN0,0:INIT:CLS
40 '♣♣♣♣ MENU ♣♣♣♣
50 CONSOLE:LINE(0,5)-(39,5),"−"
60 LOCATE6,0:COLOR6:PRINT"♣♣♣♣ ﾊﾟｿｺﾝ ｱﾄﾞﾚｽﾌﾞｯｸ ♣♣♣♣"
70 COLOR4:LOCATE0,2:PRINT"DATA INPUT --- [1]  SEARCH   --- [2]
80 COLOR5:LOCATE0,4:PRINT"MAKE CMT   --- [3]  LOAD CMT --- [4]
85 COLOR3:LOCATE20,6:PRINT"SELECT COMMAND [ ]"
90 I=1
100 LOCATE36,6:REPEAT:Z$=INKEY$(1):UNTIL Z$<>"" AND (Z$="1" OR Z
$="2" OR Z$="3" OR Z$="4")
110 CONSOLE8,16:ON VAL(Z$) GOSUB"INPUT","SEARCH","SAVE","LOAD"
190 CONSOLE:GOTO 100
200 LABEL"INPUT":I=EF+1
210 LOCATE0,8:PRINT"NAME :END ﾃﾞ ｵﾜﾘ"
220 LOCATE0,10:PRINTCHR$(5):LOCATE0,10:INPUT"NAME :",NAM$
230 WHILE NAM$<>"END"
240 LOCATE0,13:PRINTCHR$(5):LOCATE0,13:INPUT"ADRES:",ADRS$
250 LOCATE0,17:PRINTCHR$(5):LOCATE0,17:INPUT"TEL. :",TEL$
260 LOCATE0,22:PRINT"OK ? (Yes or No)"
270 LOCATE18,22:REPEAT:Z$=INKEY$(1):UNTIL Z$="Y" OR Z$="N":CREV
280 IF Z$="Y" THEN NAM$(I)=NAM$:ADRS$(I)=ADRS$:TEL$(I)=TEL$:I=I+
1:GOTO220
290 IF Z$="N" GOTO220
300 WEND
310 EF=I:CLS:CONSOLE
320 RETURN
350 LABEL"SEARCH"
360 I=1
370 CLS:LOCATE0,8:COLOR3:INPUT"NAME :",SEARC$
380 FOR I=1 TO EF
390 IF SEARC$=NAM$(I) GOSUB"PRINT":GOTO410
400 NEXT
405 LOCATE0,11:COLOR2:PRINT"ｶﾞｲﾄｳｼｬ ﾅｼ "
410 LOCATE0,22:COLOR3:PRINT"ﾂﾂﾞｹﾏｽｶ (Yes or No)"
420 LOCATE20,22:REPEAT:Z$=INKEY$(1):UNTIL Z$="Y" OR Z$="N"
430 IF Z$="N" THEN CLS:RETURN
440 GOTO370
500 LABEL"PRINT"
510 LOCATE0,11:PRINT"ADRES:";ADRS$(I)
520 LOCATE0,15:PRINT"TEL  :";TEL$(I)
530 RETURN
600 LABEL"SAVE"
610 OPEN "O",#1,"CAS:TEL"
620 FOR I=1 TO EF
630 WRITE#1,NAM$(I);ADRS$(I);TEL$(I)
```

```
640 NEXT
650 CLOSE
660 RETURN
700 LABEL "LOAD"
710 I=1:OPEN"I",#1,"CAS:TEL"
720 IF EOF(1)=-1 GOTO 750
730 INPUT#1,A$,B$,C$
732 NAM$(I)=A$:ADRS$(I)=B$:TEL$(I)=C$
740 I=I+1:GOTO720
750 EF=I-1:CLOSE:RETURN
```

入力部分の基本は，「テレビ24時間」とほとんど同じです。全体の構成は，「メニュー形式」になっており，このメニューからどのコマンドへも移れるようになっています。230行に，WHILEがありますが，これは，REPEAT～UNTILと同じようなループを作るステートメントです。300行のWENDと対になって使われます。REPEAT～UNTILは，条件の判断をループの終わりであるUNTILで行なうのに対し，WHILE～WENDでは，ループの入口に書くWHILEで判断するところが違います。この違いは，REPEAT～UNTILは少なくとも１回はループを通るのに対し，WHILE～WENDでは，１回もループを通らない場合もあります。

〔図８―４〕

ループの終りで判断しているため、最低１回はループを通る

最初から条件が成り立たないときは、１度もループを通らない

このプログラムの全体は，４つのサブルーチンで組み立てられています。データを入力する部分，氏名から住所と電話番号をさがす部分，そして，ファイルへの書き込みと読み出しの部分になっています。600行から660行がテープへの書き込みの部分で，配列に記憶されているデータを順番に書き込みます。ここでは，現在いくつのデータが入っているかが，変数EFに代入されています。

ファイルからの読み出しを行なうのが700行からの部分で，710行でOPEN“I”として，読み出しのためにファイルOPENすることを指定しています。このとき，セットされたテープに，いくつのデータが入っているのかわかりませんから，FOR～NEXT文を使うことはできません。そこで，720行で，EOFという関数を使って，ファイルから最後のデータが読み出されたかどうかを調べています。この関数の値は，ファイルの最後を読み込んだときに－１になります。言いかえれば，ファイルへの書き込みのときにファイルがCLOSEされた時点で，－１がテープに書き込まれるわけです。従って，この値が－１になれば，もう読み出すデータは残っていないこと

になりますから，ファイルを閉じて，次の処理へ，この場合では「メニュー」に戻ることになります。この方法を使うことで，ファイルに記録されるデータの数があらかじめわかっていなくても，最後まで正しく読み出せるわけです。

●255文字を一気に読み込む

ファイルからのデータ読み出しには，もう１つ， LINEINPUT＃，または LINPUT＃というステートメントがあります。これは，画面上の１行分を文字変数に入れるLINEINPUTと同じような働きをするもので，ファイルから255文字分のデータを一気に文字変数に読み込みます。扱うデータが多い場合，データをすべて文字変数に直してPRINT＃文で書き込んでおくと，255文字まとめて読み出すことができます。後は１つの文字変数に読み込まれたデータをもう一度分解してやれば良いわけです。

●急ぐときはこれ！

さらに，書き込み，読み出しを高速で行ないたい場合，DEVO$, DEVI$を使うと便利です。DEVO$, DEVI$は，256バイト（256文字）分のデータをまとめて書き込み，読み出しを行なう命令です。

```
DEVO$ " CAS: " , n, A$, B$
DEVI$ " CAS: " , n, A$, B$
```

と書きます。nはレコード番号（ファイル番号とは別のもの）です。A$, B$には128バイトずつ，合計256バイトの文字を入れて置き，これをレコードとします。ファイルディスクリプタに「MEM:」や「EMM:」を使うとランダムファイルとして使うことができます。この場合，レコード番号によって指定したレコードから読み出しが行なわれます。〔プログラム８－７〕は，「MEM:」を使ってランダムファイルを使えるようにした例です。

〔プログラム８－７〕

```
10 OPTION SCREEN2:INIT"MEM:"
20 A$=STRING$(128,"A"):B$=A$
30 C$=STRING$(128,"B"):D$=C$
40 E$=STRING$(128,"C"):F$=E$
50 DEVO$"MEM:",1,A$,B$
60 DEVO$"MEM:",2,C$,D$
70 DEVO$"MEM:",3,E$,F$
80 INPUT"ﾚｺｰﾄﾞ NO.",REC
90 DEVI$"MEM:",REC,X$,Y$
100 PRINT X$;Y$
110 GOTO 80
```

DEVO$, DEVI$は，大変高速な処理ができ，しかもランダムアクセスが可能です。必ず１レコードの長さを256バイトにそろえなければなりませんから，プログラム中でこの処理を行なう必要があります。またこのステートメントを使って記録されているときには，FILESを取ることはできません。FILES命令で表示されるような一切のもの（ディレクトリ）を記録しないことも，この高速性に一役買っているわけです。

なお，カセットテープを使った場合でも，シーケンシャルに記録されているレコード番号をさがせばよいわけですから，テープを巻き戻してあれば，レコード番号を指定することにより，ランダムアクセス的な処理ができます。

〔プログラム 8 - 8〕

```
10 '
20 A$=STRING$(128,"A"):B$=A$
30 C$=STRING$(128,"B"):D$=C$
40 E$=STRING$(128,"C"):G$=E$
50 DEVO$"CAS:",1,A$,B$
60 DEVO$"CAS:",2,C$,D$
70 DEVO$"CAS:",3,E$,F$
80 REW:INPUT"ﾚｺｰﾄﾞ No.",REC
90 DEVI$"CAS:",REC,X$,Y$
100 PRINT X$;Y$
110 GOTO 80
```

●大量のデータ処理は？

ビジネス用のプログラムなどで，グラフィックを使用しない場合は「MEM: 」を使うことで最大190レコードのデータを扱えます。当然これらのデータは，メインメモリには関係しませんから，大量のデータを処理することが可能です。作業の終了時には「MEM: 」に入っているデータをカセットへ移しておけば，次からは，最初にこのデータを「MEM: 」に入れ直すことによって，高速データ処理ができるようになります。〔プログラム 8 - 9〕はこの例で，データとして33から222までのアスキーコードの文字を，256文字分つなげたものを仮のデータとしています。この仮データを作る部分（30から60行）を実用的なものに作りかえることで応用できます。

また，SCREEN文によって「MEM:」の状態を画面に表示させていますから，DEVO$でデータが書き込まれる様子が見られるようになっています。実用として使う場合には，プログラム中に「SCREEN」とだけ入れておけば，画面には表示されません。70行でレコード番号を入力すると，そのレコードのデータが表示されます。ここで－1を入力すると，120行に移り，DEVO$とDEVI$を交互に使って「MEM:」の内容をカセットテープに移します。200行以降のプログラムを実行して，この逆の働きをさせることによって，カセットテープのデータを「MEM:」に移し，ランダムファイルとして使うことができるようになります。

〔プログラム 8 - 9〕

```
10 SCREEN0,0:CLS4
20 CLS:OPTIONSCREEN2:INIT"MEM:"
30 FOR I=33 TO 222
40 A$=STRING$(128,CHR$(I)):B$=A$
50 DEVO$"MEM:",I-32,A$,B$
60 NEXT:SCREEN
70 INPUT"ﾚｺｰﾄﾞ No.(1 ｶﾗ 190) -1 ﾃﾞ CMT ",REC
80 IF REC=-1 GOTO 120
90 DEVI$"MEM:",REC,X$,Y$
100 PRINTX$;Y$
110 GOTO 70
120 FOR I=1 TO 190
130 DEVI$"MEM:",I,A$,B$
140 DEVO$"CAS:",I,A$,B$
150 NEXT
160 END
200 FOR I=1 TO 190
210 DEVI$"CAS:",I,A$,B$
220 DEVO$"MEM:",I,A$,B$
230 NEXT
240 END
```

8-6 ファイルディスクリプタとINIT文

●ファイルディスクリプタってなに

ここでは、ファイルディスクリプタについて詳しく説明してみましょう。このことばは前の《8-5》でも出てきました。

ファイルディスクリプタは，プログラムやデータを外部機器に出し入れするときに，その外部機器（外部デバイス）を指定する文字定数です。この文字定数には次のようなものがあります（表8-1）。

表8-1

ファイル ディスクリプタ	デバイス	モード	使える命令の例
SCR:	画　面	0	PRINT#　WRITE#
CRT:	〃	〃	
KEY:	キーボード	I	
LPT:	プリンタ	0	
CAS:	カセットテープ	I/0	PRINT#　WRITE# SAVE, LOAD DEVI$, DEVO$
MEM:	グラフィックメモリ	I/0	〃
EMM0: ～ EMM9:	外部メモリ	I/0	〃

この表からもわかるように，実際にプログラムやデータを出し入れすることができるのは，"CAS:"，"MEM:"，"EMM 0 :～EMM 9 :" です。

SCR:とCRT:は，どちらも画面を示しますが，31以下のアスキーコードを持つ「コントロールコード」の扱いが異なります。たとえば，A$に0から31までのアスキーコードのコントロールコードが入っている場合，

```
OPEN" 0 ",# 1,"CRT:"
PRINT# 1,A$
```

とすると，コントロールコードの内容は実行されず，コントロールコードのキャラクタが表示されます。一度CLOSEしてから，

```
OPEN " 0 " ,# 1, "SCR: "
```

として同じことを行なうと，コントロールコードの内容が実行されます。実際には画面にファイルを作ることはないのですが，擬似的に使って，ファイルの書かれたフォーマットを見ることができます。

たとえば，

```
A$="A":B$="B":C$="C"
```

として，

```
OPEN" 0 ",# 1,"CRT:"
```

として，

PRINT＃１, A$; B$; C$

とすると,

　　ABC

と表示されます。これは, PRINT＃文のセパレータの働きによって, ３つの文字変数が連続して書かれてしまいます。これがカセットテープに書かれているとすると, このファイルを読み込もうとするときに,

　　INPUT＃１, A$, B$, C$

としても, A$に"ABC"が読み込まれてしまい, エラーになるわけです。

　ところが,

　　WRITE＃１, A$, B$, C$

とすると,

　　"A", "B", "C"

となり, 読み込みがうまくいきます。

　このようにデバイスへのプログラムやデータの入出力は, ファイルディスクリプタを書きかえるだけで, どのデバイスもまったく同じ感覚で使うことができます。

　なお, ディスクBASICの場合, これ以外にディスクドライブの番号を示す０: ～3: が使用できるようになります。ファイルディスクリプタは文字定数ですから, コロン（：）を付けるのを忘れないようにしましょう。

●外部デバイスの初期化を忘れずに

　外部デバイスにプログラムやデータを入出力するには, INIT文であらかじめそのデバイスを初期状態にしておかなければなりません（イニシャライズ)。通常INITを実行すると,「SCR：」,「CRT：」のイニシャライズが行なわれ,「COLOR 7, 0 : CGEN : CFLASH : CSIZE : CREV : PALET : PRW : WINDOW : CONSOLE : SCREEN 0, 0, 0」が実行されたのと同じ状態になります。INIT文の後にファイルディスクリプタを付けることで, 希望のデバイスをイニシャライズできます。なお,「KEY：」と「LPT：」はエラーになり, 何も実行されません。ファイルディスクリプタを付けたINIT文をダイレクトに実行すると,

　　Are you sure ? (y or n)

というメッセージが表示され, 確認を取るようになっています。これは, イニシャライズが行なわれると, それまでの内容がすべて消去されるためです。特に, INIT"CAS: とした場合, テープが巻き戻されてから本体の右にある「WRITE」のランプがつき, テープの内容が消去されるのがわかります。このメッセージは, プログラム中にINIT文を置いた場合には表示されずに, ただちにイニシャライズが実行されてしまいますので注意しましょう。

●プログラム作成中にSAVEしたいときは……

　データの入出力の場合, OPEN文でファイルディスクリプタを指定してデバイスを決めますが, SAVEやLOADの場合に何も指定しないと,「CAS：」を指定したことになります。プログラム作成中で, 一時的にプログラムをSAVEしておきたい場合, グラフィックメモリを使うと便利です。

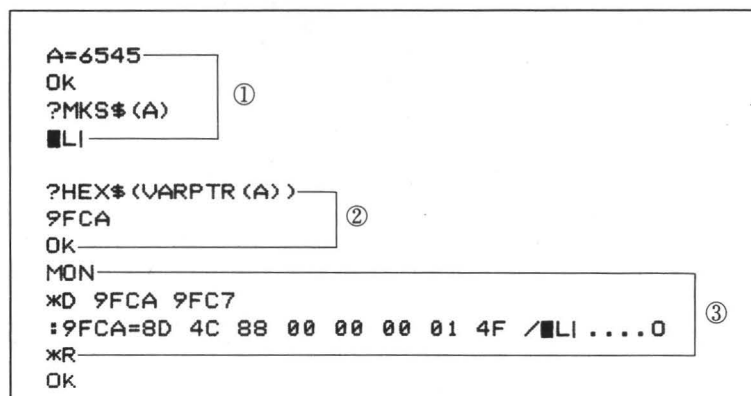
SAVE "MEM:

として指定します。DEVICE文を使うとSAVEやLOADを行なうデバイスをあらかじめ決めておけます。

DEVICE "MEM:

DEVICE "EMM1:

のようにしておくと，それ以後，ファイルディスクリプタを付けないSAVE，LOADは，DEVICE文で指定したデバイスに対して行なうことになります。

なお，グラフィックメモリや，外部メモリにSAVEしたものには，VERIFY（LOAD？）は行なえず，Device I/O errorになってしまいます。また，グラフィックメモリを使った場合，グラフィックをクリアするCLSnの文を実行すると消去されますから注意してください。

8-7 カセットテープの節約

●数字の文字列変換

ファイル処理によって，たくさんのデータを外部ファイルに記録できますが，数値データを大量に記録するときには思いのほか時間がかかるものです。特にデータの桁数が多い場合は大変時間がかかります。これは，ファイルに記録するのに，数字をそのまま書き込んでいるためです。

パソコンの内部では，数値データはそのまま扱われているわけではなく，前にも説明したとおり，パソコンにとって都合のよい2進数の形で記憶されています。たとえば，有効数字8桁の単精度の数値は，40桁の2進数，つまり5つの16進数によって記憶され，倍精度の数値は7つの16進数，指定された数値については，2つの16進数で記憶されています。ですから，どんな桁数の多い数値でも，単精度なら5文字，倍精度なら7文字，整数化されていれば2文字の文字列に置きかえられることになります。このコンピュータ内部表現を使ってファイルに記憶すると，ファイルへの書き込みスペースが短くなりますから，書き込む速度も早くなります。

画面8-2

```
A=6545
OK
?MKS$(A)
■LI

?HEX$(VARPTR(A))
9FCA
OK
MON
*D 9FCA 9FC7
:9FCA=8D 4C 88 00 00 00 01 4F /■LI....O
*R
OK
```


数値を内部表現の文字列に変換するのが，

MKI$(X)：整数値を2文字の文字列に
MKS$(X)：単数度値を5文字の文字列に
MKD$(X)：倍精度値を7文字の文字列に

という関数です。試してみましょう(画面8-1①)。5文字になるところが3文字しか表示されません。

この理由を調べるために，実際に変数Aの値が記録されている様子を見てみましょう。まず変数Aの値が入っているアドレスを調べます。そのための関数が，VARPTR(A)です。こうすると，変数Aの記憶されているアドレスが求められますから，これをHEX$によって16進数に直します（画面8-2②)。そして，モニタのDコマンド（MON）を使って，このアドレスから8バイトを表示してみました（モニタのコマンドについては《9-1》参照)。すると，はじめの3文字はMKS$によって表示されたものと一致し，残り2文字は00なので表示されなかったこと

がわかりました（画面8－2③)。BASICに戻ってから「LEN (MKS$ (A))」としてみると，5が表示されることからも確かめられます。

さらに，Aの値をもっと大きな値や，桁数の多い小数にしていても，この関数で求められる文字列の長さは常に5文字であることが確かめられます。

●何と速度が倍！

それでは，実際にファイルへの記録がどのくらいはやくなるかを試してみましょう。サンプルデータとしては乱数を使い，1000個の数値データとしてカセットテープに書き込んでみます。数値をそのまま書き込むのが〔プログラム8-10〕です。

〔プログラム8-10〕

```
10 INIT:WIDTH40:CLS
20 RANDOMIZE10
30 OPEN"O",#1,"CAS:
40 FOR I=1 TO 1000
50 PRINT #1,RND(1)
60 NEXT
70 CLOSE
80 END
```

実際にテープを入れ，テープカウンタを000にしてこのプログラムを動かしてみると，カウンタの値が41くらいで終わります。

〔プログラム8-11〕

```
10 INIT:WIDTH40:CLS
20 RANDOMIZE10
30 OPEN"O",#1,"CAS:
40 FOR I=1 TO 1000
50 PRINT #1,MKS$(RND(1));
60 NEXT
70 CLOSE
80 END
```

〔プログラム8-11〕では，MKS$によってデータを内部表現の5文字に変換してから書き込んでいます。この場合，同じようにして調べてみると，カウンタの値は19くらいで終わりますから，約2分の1の時間で済んだことになり，かなりの短縮になります。

なお，20行の「RANDOMIZE」はどちらのプログラムも同じ乱数を発生させるためのものです。オペランドを省略すると毎回違った乱数が，数値や式をつけるのと同じ順序の乱数を発生させることができます。

●もとの数値にもどすには

この方法で作ったファイルから読み出されるのは，内部表現に変換された文字列ですから，今度はこれをもとの数値に変換し直さなければなりません。MKI$, MKS$, MKO$に対応するこれらの関数は，次のようになっています。

```
A$=MKI (A) ←→A=CVI (A$)
A$=MKS (A) ←→A=CVS (A$)
A$=MKD (A) ←→A=CVD (A$)
```

これを使った読み出しの例が〔プログラム8-12〕です。

〔プログラム8-12〕

```
10 INIT:WIDTH40:CLS
20 OPEN"I",#1,"CAS:
30 FOR I=1 TO 1000
40 A$=INPUT$(5,#1)
50 PRINTCVS(A$):NEXT
60 CLOSE
```

これは，〔プログラム8-11〕で作ったファイルを読むものです。〔プログラム8-11〕は，50行のセミコロンによって，テープ上には連続した文字列で書き込まれています。〔プログラム8-12〕では，この連続した文字列を，40行のINPUT$を使って5文字ずつ区切って読み込み，50行でCVSを使って数値に直しています。

ファイル処理は，使う人にとってただ終るのを待っているだけの作業ですから，このような方法を使って少しでも短縮することで，能率のよいデータ処理ができるようになります。ビジネス用のプログラムにはぜひ活用してみてください。

らんだむめも

グラフィックRAMのもう1つ便利な顔

プログラムを作成中に他のプログラムを参考にしようとするとき，便利な方法を紹介しましょう。ただしこれはX1のオプションのグラフィックRAM（CZ-8GR）が組み込まれていないと使えません。X1では，グラフィックRAMを2つの用途に使うことができます。1つは文字どおり点を打ったり，線を引いたりして画面をつくるグラフィックメモリとしての使い方です。もう1つは，プログラムやデータを記録する外部メモリとして使う方法です。

仮りにAというプログラムを作っている途中で，Bというプログラムの内容を参考にするとしましょう。プリンタを持っている方はBのリストを打ち出しておいて見ればよいのですが，そうでない場合，Aを一度テープに記録しておいてから次にBを**LOAD**し，Bの内容をリストを見て確認した後，またAを**LOAD**するという少々手間のかかる作業をしなければなりません。このようなとき，グラフィックRAMを外部メモリとして使うと，手間もかからず，しかもはやく同じ作業を行なうことができます。

グラフィックRAMをカセットテープと同じように使って，プログラムAを**SAVE**してしまうのです。こうしておいてこんどはカセットテープからプログラムBを**LOAD**してリストを見た後，またグラフィックRAMからAを**LOAD**してしまうのです。グラフィックRAMへの**LOAD**，**SAVE**は大変高速ですから，これは非常に便利な使い方です。

ただし，ここで注意しなければいけないことは，この操作は画面が横40文字のモードになっていないと使えないことです。

また，プログラムBにグラフィックを使った部分があると，このプログラムを**RUM**させたときにグラフィックRAMに**SAVE**されているプログラムAが消されてしまうことがあるという点です。ですから，プログラムBの内容を参照するだけであれば，リストを見るだけにとどめておきましょう。また，この技法はX1ならではの，いわば秘技で，プログラムやデータの一時的記録には大変便利ですからうまく使いこなしてください。

9

機械語も使えます

9-1 機械語のはなし

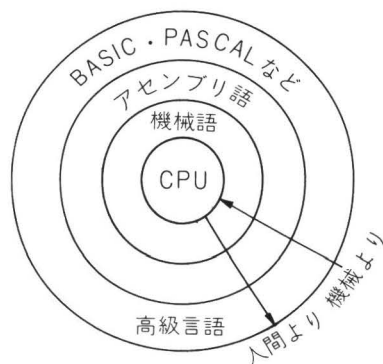
●コンピュータ言語の輪

ここまで，X１のBASICについて説明を進めてきました。その中で，コンピュータは，０と１の信号の組み合わせで動作しているということにも何度かふれてきました。

この機械語は，コンピュータ言語というよりも，機械を動かすためのコードのようなもので，私たち人間にとっては大変難解なものです。実際，コンピュータがこの世に誕生したばかりのころは，この機械語によってプログラミングしていましたので，１つのプログラムを完成させるのに莫大な労力と時間が必要でした。そのためにまず考えられたのが，機械語の動作をある程度人間が理解できるように表現する**アセンブリ言語**です。アセンブリ言語でプログラミングされたものは，**アセンブラ**というプログラムの働きによって機械語に直すことができます。しかし，アセンブリ言語も，機械語の命令に１対１で対応するよう作られているため，大きなプログラムを作ろうとする場合にはやはり，多くの労力と時間が必要でした。また，コンピュータの中枢であるCPU（中央演算処理装置）が異なればそれを動かすための機械語が異なるため，アセンブリ言語もCPUごとに違ったものが必要でした。

〔図９－１〕

コンピュータをさらに使いやすく，理解しやすいようにするために考えられたのが「BASIC」をはじめとする高級言語と呼ばれるものです。これらの高級言語の発達によって，パソコンも広く普及してきたわけです。

これらの言語を仲介としたCPUと人間との関わり合いを表わしたのが，図９－１の「言語の輪」です。この図でもわかるように，人間にとって最も理解しやすいのがBASICなどの高級言語，CPUにとって理解しやすいのが機械語やアセンブリ言語ということになります。

●機械語の長所と短所

私たちが，コンピュータに何か仕事をさせるときにも，BASICを使えばほとんどのことができます。

実はこのBASICも，機械語プログラムの集まりでできており，入力された命令について必要な機械語サブルーチンを次々と呼び出すことで働いています。BASICの命令を機械語に翻訳するというのは，この作業のことなのです。BASICは，翻訳作業の手間があるために，実行速度に限りがあり，プログラムの内容によっては大変遅いと感じられることもあります。

これに対して，機械語はCPUを直接動かすものですから，実行速度に関しては比較にならない高速性を得ることができます。また，プログラム全体が機械語で作られているものなら，BASIC自体が不要になりますから，コンピュータのメモリもフルに使うことができます。このように書くと良いことずくめのようですが，前にも書いたように，人間にとって大変理解しにくいと

いう短所もあります。また，BASICでは１つの命令で済むような処理でも，機械語では何ステップものプログラムを組まなければならない場合が多く，一度作ってしまった機械語のプログラムは後から簡単に修正することができない，という大きな短所があるのです。

●機械語とはどんなものか

このような長所と短所を考えると，パソコンのプログラムで機械語を使うのは，特に速度を重要視する場合と考えてよいでしょう。Ｘ１に使われているZ80AというCPUは，158種の機械語の命令を持っています。これらの命令については１つ１つ説明しているとそれだけで１冊の本になってしまいます。興味のある方は他によい本もたくさん出版されていますので，そちらを参照していただくことにして，ここでは機械語がどのようなものかを見るだけにとどめましょう。非常によく似た内容の命令をBASIC，アセンブリ言語，機械語で書き表わしてみながら，それぞれの違いを見ていくことにします。

たとえば，BASICで「A＝1」とすることを考えてみましょう。機械語では，Aのような変数はありませんから，メモリに数値を書き込むことになります。このメモリの番地をF008H（Hは16進数を表わす）書き込むデータを2EHとすると，アセンブリ言語では，

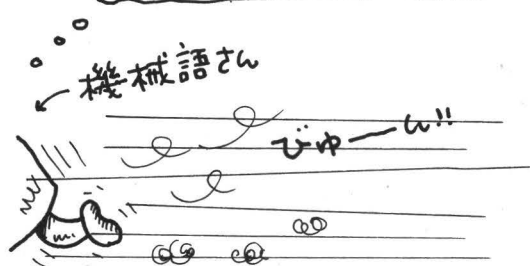
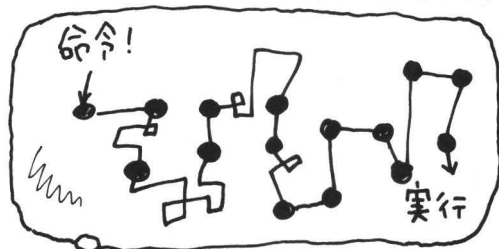
```
LD A, 2EH
LD (F008), A
```

と書き表わされます。Aというのは，CPUが持っているレジスタの１つで，主に計算の結果を保持する働きを持っています。レジスタにはこの他に，F，B，C，D，E，H，Lの他，22個のものがあります。ここで行なっていることは，このAレジスタに2EHというデータを入れ，次にF008Hという番地（アドレス）のメモリに，このAレジスタの値を書き込む，というものです。この作業を機械語で表現すると，次のようになります。

```
LD A, 2EH        :3E 2E
LD(F008H), A     :32 08 F0
```

このように，機械語の命令はそれぞれがとても単純な作業を行なうため，まとまった処理を行なわせようと，いくつもの命令を組み合わせていかなくてはならないため，プログラムの量が多くなるわけです。また，BASICと違って，行番号やラベルはなく，直接アドレスで指定することになります。

機械語を使ううえで特に注意しなくてはいけないのは，機械語の命令が，１つまたは２つの16進数の組み合わせでできているという点です。ですから，もし，１文字でも打ち間違えると，その命令はまったく別の命令を表わすことになってしまいます。BASICでは，エラーメッセージで警告してくれますが，機械語ではこのような機能はなく，そこに書かれているとおりの内容を実行してしまいます。

たとえば先ほどの例で，32 88 F0と打ち込むべきところを，C3 08 F0と打ってしまったとす

ると，これは，F008番地へジャンプせよ，という機械語を意味することになり，ほとんどの場合コンピュータは，コントロール不能の状態(暴走状態)になってしまいます。X 1では，BASICやモニタプログラム自身も，書きかえのできるRAMに書き込まれていますから，最悪の場合これらのプログラムが破壊されることがあります。こうなったら，一度電源を切ってやり直すしかありません。もちろん打ち込んであったプログラムやデータもすべて消えてしまいます。

●機械語を使うには

機械語は，今までに説明したような短所を持っているかわり，その速度には目を見張るものがあります。ここでは実際に機械語を扱うための方法を説明しましょう。

X 1のBASICには機械語のプログラムを作ったり，実行したりするための機械語モニタが内蔵されています。モニタにコントロールを移すには，BASICのコマンドレベルから

MON ↵

とします。すると画面上には，＊（アスタリスク）に続いてカーソルが表われ，コントロールがモニタに移ったことを示します。このモニタから使えるコマンドを，先ほどの例を使って説明しましょう。

＊M（メモリセット）

メモリの内容を書きかえる，つまりメモリへの書き込みを行ないます。＊Mに続いて，書き込みをはじめたいアドレスを16進4桁で入れます。先ほどのプログラムをF000Hから書き込んでみましょう。

＊M　F000 ↵

：F000＝[0]0

と表示されますから，3Eと書きかえます。↵キーを押すと次のアドレスとデータが表示されますから，同じように書き込みます。これを，次のようにして書き込むこともできます。

＊M　F000 ↵

：F000＝3E　2E ↵

：F002＝[0] 0

一般には，

：アドレス＝データ␣データ␣データ␣……（␣は1文字文のスペース）

の形式で，16バイト分一度に書き込むことができ，それ以降は無視されます（1バイトというのは，2桁の16進数にあたります)。書き込みを続けましょう。

：F002＝32　08　F0　C9

C9は，BASICのRETURNにあたる機械語で，この機械語プログラムが呼び出されたところに戻ります。C9まで書き込んだら，[SHIFT]＋[BREAK]でモニタのコマンドレベルに戻ります。機械語プログラムの打ち込みは，このように一見無意味な数字の連続ですから，間違いのないよう，注意する必要があります。

＊D（ダンプ）

BASICで言えばLISTに相当するもので，メモリの内容を表示します。今書き込んだ部分をダンプし，間違いがないか確認してみましょう。

```
*D F000 F007 ⏎
```

とすると，

```
:F000=3E 2E 32 08 F0 C9 00 00 />.2.▒/..
```

と表示されます。F000はダンプを開始するアドレス，F007は終了アドレスです。終了アドレスを省略すると，開始アドレスから128バイト（16行）ダンプします。“／”（スラッシュ）から後の8文字は，メモリの内容をアスキーコードとした文字です。1FH (31) 以下のコントロールコードについては，“.”（ピリオド）が表示されます。

＊P（プリンタスイッチ）

Pコマンドと後から説明するFコマンドの出力を画面とプリンタどちらにするかを切りかえます。BASICからMONコマンドでモニタに入ったときは，出力が画面となります。その後Pコマンドを実行するとプリンタに切りかわり，もう一度Pコマンドを実行すると再び画面になります。以後，実行するたびに，交互に切りかわります。

＊G（ゴーサブ）

＊Gに続いて，16進4桁のアドレスを付けると，そのアドレスからの機械語サブルーチンを呼び出します。このプログラムを実行するには，

```
*G F000 ⏎
```

とします。すぐに，コマンド待ちに戻れば完了です。もし，なかなか戻らないときは，打ち込みのあやまりが考えられますから，すぐに本体後ろのリセットスイッチを押してください。BASICのコマンド待ちに戻ったら，もう一度モニタに移り，Dコマンドで間違いがないか確めてください。なお，Dコマンドで表示されている内容をBASICのスクリーンエディットと同じようにカーソルを動かして修正することもできます。

うまく動いた場合は，DコマンドでF008番地の内容を調べると，2Eになっているはずです（図9－2）。

〔図9－2〕

```
:F000=3E 2E 32 08 F0 C9 00 00 />.2.▒/..  } 実行前
:F008=00 00 00 00 00 00 00 00 /........  }

:F000=3E 2E 32 08 F0 C9 00 00 />.2.▒/..  } 実行後
:F008=2E 00 00 00 00 00 00 00 /........  }
```

＊T（トランスファ）

あるアドレスの内容を他のアドレスに移動させます。一般的な書き方は，

```
*T アドレス1␣アドレス2␣アドレス3
```

とし，アドレス1からアドレス2までの内容をアドレス3を先頭とするアドレスに転送します。このプログラムを，F800Hへ転送してみましょう。

*T　F000　F005　F800

実行前と実行後のF800Hからの内容を見てると，次のようになっています（図9-3）。

〔図9-3〕

```
:F008=00 00 00 00 00 00 00 00 /........ 実行前

:F800=2E 3E 32 08 F0 C9 00 00 /.>2.▒/.. 実行後
```

ここで気を付けなければいけないのは，BASICのRENUMBERと違い，ジャンプ先やデータをセットするアドレスはもとのままということです。図9-3を見るとAレジスタの内容を書き込むアドレスを表わす，4，5バイト目が，08　F0のままになっています。この場合なら，特に問題は生じませんが，これがジャンプアドレスだった場合には，大変なことになります。つまり，Tコマンドを実行しても，もとのアドレスの内容は変化せずに残っているからです。

＊S（セーブ）

BASICのSAVEと同じですが，セーブするアドレスを指定する必要がありますので，次のように書きます。

＊S　セーブ開始アドレス　セーブ終了アドレス　実行開始アドレス：ファイルネーム

セーブ開始アドレス，終了アドレスは，このプログラムの場合，F000，F005になります。実行開始アドレスをF000とすれば，このプログラムをロードしたときに，自動的に実行されます。自動的に実行したくないときは1003Hを指定すれば，モニタのコマンドレベルに入ります。また，14C2Hにしておくと BASICのコマンド待ちにジャンプしますから，BASICから機械語プログラムをサブルーチンとして使うようなときに，BASICプログラム中でLOADM文を使うことができます。

＊V（ヴェリファイ）

BASICのLOAD?，VERIFYと同じで，「"：”」（コロン）に続けてファイルネームを指定できます。正しくセーブされていればモニタのコマンド待ちに，正しくなければ「ERR?」と表示して止まります。

ディスクを使っていない場合，S，L，Vコマンドのデバイスはカセットテープになります。これは，BASICのDEVICE文で他のデバイスを指定してあっても変わりません。

＊L（ロード）

BASICのLOAD文と同じですが，ロード先頭アドレスを指定することができます。指定しない場合，Sコマンドで指定したとおりのアドレスをロードします。ファイルネームの扱い方はBASICのLOAD文と同じです。使い方は次のようになります。

＊L　アドレス：ファイルネーム
＊L　アドレス
＊L　：ファイルネーム

Sコマンド，Lコマンドを SHIFT ＋ BREAK キーで止めたり，Lコマンドで読み込みエラーが出ると「ERR?」と表示されます。

＊F（ファインド）

SEARCH命令に相当するものです。さがすのは，16進数2桁のデータの並びで，

＊F　開始アドレス␣終了アドレス␣データ データ　…

とします。例として，BASICの「NEXT」をさがしてみましょう。この4文字のアスキーコードは16進数で，4E 45 58 54ですから，

```
*F  0000  AFFF  4E  45  58  54  ↵
```

とします。すると，画面9-1のように，736BH（Hは16進数を表わす記号）と7574Hの2つのアドレスからこのデータの並びがあることがわかります。試しに，736BHからDコマンドでダンプしてみると，BASICのエラーメッセージが入っていることがわかります（画面9-2）。ダンプしたときの右側の8文字の部分では，このようにメッセージなどを直接文字で見ることができます。このようなメッセージをメモリに書き込むときには，Mコマンドで次のようにすることができます。

〔画面9-1〕

```
:736B=4E 45 58 54 /NEXT
:7574=4E 45 58 54 /NEXT
```

〔画面9-2〕

```
:736B=4E 45 58 54 20 77 69 74 /NEXT wit
:7373=68 6F 75 74 20 46 4F 52 /hout FOR
:737B=00 53 79 6E 74 61 78 20 /.Syntax 
:7383=65 72 72 6F 72 00 52 45 /error.RE
:738B=54 55 52 4E 20 77 69 74 /TURN wit
:7393=68 6F 75 74 20 47 4F 53 /hout GOS
:739B=55 42 00 4F 75 74 20 6F /UB.Out o
:73A3=66 20 64 61 74 61 00 49 /f data.I
:73AB=6C 6C 65 67 61 6C 20 66 /llegal f
:73B3=75 6E 63 74 69 6F 6E 20 /unction 
:73BB=63 61 6C 6C 00 4F 76 65 /call.Ove
:73C3=72 66 6C 6F 77 00 4F 75 /rflow.Ou
:73CB=74 20 6F 66 20 6D 65 6D /t of mem
:73D3=6F 72 79 00 55 6E 64 65 /ory.Unde
:73DB=66 69 6E 65 64 20 6C 61 /fined la
:73E3=62 65 6C 00 53 75 62 73 /bel.Subs
```

```
*M  F000
F000= ;A␣ ;B␣ ;C
```
（␣はスペースをあける）

16進データのかわりに，セミコロンに続いて直接文字を書きます。ダンプしてみると，

```
*D  F000  F007
F000=41  42  43～／ABC～
```

となっており，16進データに変換されて書き込まれています。

＊R（リターン）

モニタを呼び出したシステムプログラムへ戻ります。BASICからMONでモニタに移ったとき

は，RコマンドでBASICのコマンドレベルに戻ります。特に操作しない限り，BASICの変数やプログラムは保存されています。

●BASICの機械語に関する命令

モニタのコマンドだけでなく，BASICにも機械語に関する命令があります。これらについて説明しましょう。

PEEK, POKE

これらについては，「電子オルガン」のところで使いましたが，POKEはメモリにデータを書き込む命令，PEEKは読み出す命令です。アドレス，データとも，10進数，16進数両方使うことができ，16進数の場合は「&H」を付けます。

SAVEM

モニタで作った機械語サブルーチンをBASICからSAVEするときに使います。この場合，ファイルディスクリプタを付けることによって，デバイスを指定できます。書式は，

```
SAVEM "ファイルディスクリプタ : ファイルネーム",
先頭アドレス, 終了アドレス, 実行開始アドレス
```

とします。実行開始アドレスについては，モニタのSコマンドを参照してください。

LOADM

BASICから機械語サブルーチンや機械語データをロードします。

```
LOADM " ファイルディスクリプタ
: ファイルネーム ", 先頭アドレス, R
```

とし，先頭アドレスは省略でき，省略するとSAVEMで指定したアドレスからロードします。また，「，R」を付けておくと，ロード終了後SAVEMで指定した実行開始アドレスから自動的に実行を開始します。

PEEK@，POKE@

通常のPEEK, POKEは，メインメモリを対象としていますが，@(アットマーク)を付けると，V-RAM（ビデオRAM）に対して作用します。V-RAMのアドレスは，テキストが2000Hから37FFH，グラフィック1(青)，2(赤)，3(緑)がそれぞれ4000H～7FFH，8000H～BFFFH，C000H～FFFFHとなっています。PEEK@，POKE@でV-RAM以外のアドレスを指定するとエラーになります。

CALL（アドレス）

BASICから機械語サブルーチンを呼び出します。呼び出すサブルーチンの先頭アドレスを指定します。機械語サブルーチンの最後にはRET（C9）がなければなりません。

OUT，INP

コンピュータからデバイスにデータを送り出したり，データを受け取る出入口となるものを**ポート**と呼びます。このポートとデータのやりとりを行なうのがINP，OUTです。ポートのアドレスは，0000HからFFFFHまでありますが，X1では，2000HからFFFFHが，V-RAMにつながっています。たとえば，OUT　&H4000，&HFFを実行すると，4000HはグラフィックV-RAM青の先頭アドレスですから，画面の左上すみに青い線が描かれます。また，この位置に「A」を書いておいて，PRINT INP（&H3000）を実行すると，Aのアスキーコードである65が表示されます。0000Hから1FFFHは，各デバイスのコントロールに使われていますから，むやみにOUT命令を実行しないでください。

DEFUSR，USR

BASICから機械語サブルーチンを呼び出すには，CALL命令を使いましたが，USRを使うとデータの受け渡しができます。USR命令を使うには，まずDEFUSR命令によって機械語サブルーチンのアドレスを定義しておかなくてはいけません。

DEFUSR〔n〕＝アドレス

とし，nは0から9まで使えますから，合計10個の機械語サブルーチンを使うことができます。アドレスは，そのサブルーチンの先頭アドレスを指定します。たとえば，F000Hからある数を1000回足し算するプログラムを書いておき，DEFUSR0＝&HF000として指定しておいたとします。このときにA＝USR0（B）とすると，Bを1000回足し算した結果が機械語サブルーチンの中で計算され，Aに代入されます。また，文字に関するサブルーチンを作った場合は，A&＝USR0（B$）のような使い方もできます。つまり，USR命令は，1つの関数と考えることができるわけです。

これらの命令については，次の実用機械語サブルーチンのところで使っていますから，参考にしてください。

らんだむめも

RESET

プログラムは走っているはずなのに，X1はうんともすんともいわないしキー入力もまったく受け付けてくれない――そんな状態を経験された人はいませんか？　このような状態を「**暴走**」といいます。機械語で書いたプログラムを実行するときなどしばしば起こります。もちろんプログラムが完全でないためこういう事態を招くわけですが，かといってこのまま電源を切ってしまっては，せっかく作ったプログラムが全部消えてしまいます。なんとか「暴走」からのがれてプログラムの修正を行ないたいものです。

プログラムが暴走してしまったときは，あわてずさわがず，本体の後ろに手を伸ばし**RESET**スイッチを押してください。本体の後ろから見ると右側にある黒いスイッチです。プログラムを消すことなく実行前の状態に戻すことができます。

ここでプログラムのデバッグをするわけですが，一度暴走させてしまうとメモリの内容（特にBASICインタプリタ）がこわれて，以後の実行ができなくなる場合もあります。プログラムができたら，走らせる前にSAVEしておくことが大切です。

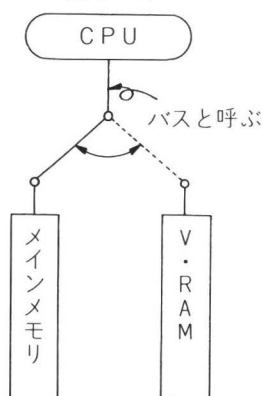
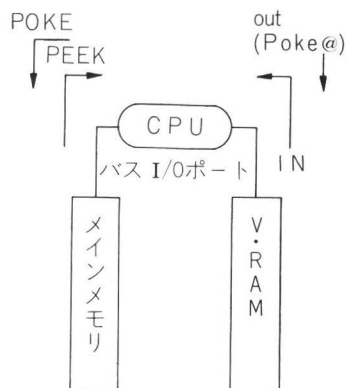
9-2 機械語のサブルーチンを作ってみよう

●グラフィック画面の塗りつぶし

グラフィックのプログラムで，バックに色を付けたいときに，COLOR文でバックカラーを指定したのでは都合の悪いときがあります。たとえば，白いバックに黒で絵を書きたいときなどです。このようなときには，LINE文のBFモードやPAINT文を使いますが，全画面を塗りつぶすにはかなりの時間がかかります。ここでは，この画面の塗りつぶしを高速で実行する機械語サブルーチンを作ってみましょう。

機械語でこのようなプログラムを作る場合，ハードウエアの構成をある程度知っておく必要があります。ここでは，画面表示を行なうV-RAMをどのようにコントロールすればよいかが分からなければ，プログラムを作ることができません。

X１では，V-RAMのコントロールを一般のパソコンとは少し違った方法で行なっています。８ビットのCPUで操作（アクセスと呼ぶ）できるアドレスは，0000HからFFFFHまでの64Kバイトです。通常，V-RAMもこのアドレスのどこかに割り当てられています。ところが，X１をはじめとする近頃のパソコンでは，細かいグラフィック表示を行なうためにグラフィックメモリの容量も多く，この方法では不都合です。X１の場合，グラフィックのV-RAMが48Kバイト，テキストのV-RAMが８Kバイトの合計56Kバイトもあります。このV-RAMにアドレスを割り当ててしまうと，メモリは８Kバイトしか残らなくなってしまいます。

多くの８ビットパソコンでは，これを解消するために，普段はCPUをメインメモリにつないでおき，V-RAMをアクセスする場合にはCPUをV-RAMにつなぎかえるという，バンク切りかえの方法が使われています（図９-４）。こうすることによって，同じアドレスをメインメモリとV-RAMで使い分けているわけです。

〔図９-４〕　〔図９-５〕

これに対しX１の場合は，V-RAMをシステムのI/Oポートに接続して使っています。I/Oポートというのは，CPUが外部の機器とデータをやりとりするときの出入口にあたる部分です。X１に使われているZ-80A CPUでは，０から255まで256個のI/Oポートしか持っていませんがI/Oのアドレスとしては，メモリアドレスと同じ，0000HからFFFFHまでを使うことがで

きます。CPUから，メインメモリとV-RAMをコントロールする様子をBASICの命令で表わすと図9-5のようになります。

X1のI/Oアドレスは次のように割り当てられています。

0000H～0FFFH	ユーザー用I/Oポート	
1000H～1FFFH	システムI/Oポート	
2000H～2FFFH	V-RAM　属性	
3000H～3FFH	V-RAM　テキスト	
4000H～7FFH	V-RAMグラフィック	(青)
8000H～BFFH	〃	(赤)
C000H～FFFH	〃	(緑)

ユーザー用I/Oポートというのは，ユーザーがX1を使って外部の機器をコントロールするときのために用意されているものです。また，オプションのRAMボード，ROMボードとフロッピーディスクのコントロールもこの部分に割り当てられています。

システムI/Oポートは，X1自身が与えられた命令を実行するためにコントロールする必要のあるものが接続され，パレットの切りかえやPCG，PSG，カセットデッキなどは，このI/Oポートからコントロールされています。2000HからのV-RAM属性は，点滅や反転，文字サイズなどの文字のタイプをコントロールしています。

I/Oポートでは，8ビットのデータを扱いますが，この属性は各ビットに次のように割り当てられています。

CSIZEH	CSIZEV	CG	CFL	CREV	G	R	B
7	6	5	4	3	2	1	0

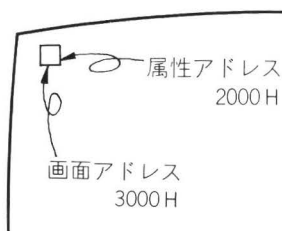
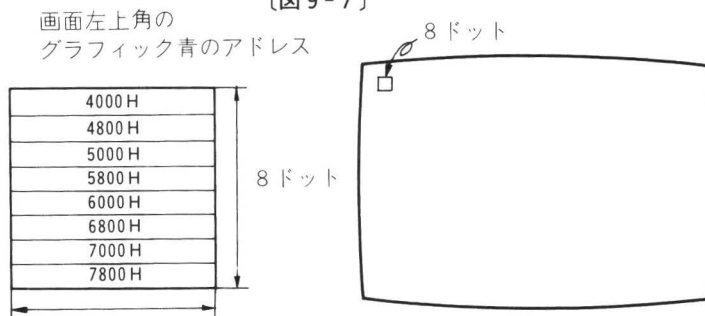
7から3までのビットは，BASICの命令と1対1で対応しています。G，R，Bのビットは，色を指定するものです。たとえば，ビット4と2，1，0を1にすると，文字の属性は点滅モードで，色は青＋赤＋緑で白になります。ビット7と6は，文字のサイズを決めるもので，ビット6が1なら縦2倍，ビット7が1なら横2倍，両方とも1なら縦横2倍になります。実際に画面上に文字を表示するには，属性を指定すると同時に，3000Hからの画面アドレスにデータを送ります。画面の左上のホームポジションは，属性アドレスが2000H，画面アドレスが3000Hになっています（図9-6）。点滅モード，色は白でAという文字を表示するためには，属性はビット構成が00010111＝16＋4＋2＋1＝23ですから，

```
OUT＆H2000, 23：OUT＆H3000, 65,
```

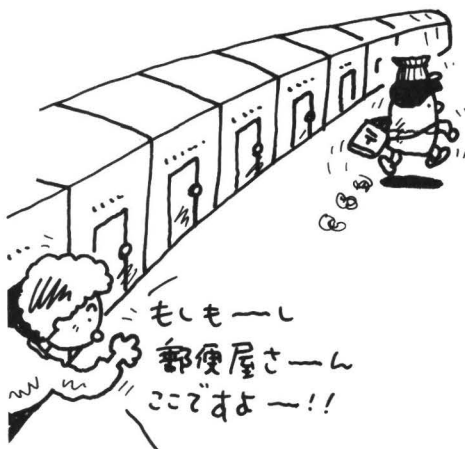
とします。属性アドレスはこのように画面アドレスと1対1に対応しています。

グラフィックのアドレスは少し複雑で，画面左上の1文字分（8×8ドット）については，図9-7のようになっています（青の場合）。このアドレスに，8ビットのデータを送ることによって，横方向8ドットをコントロールできます。SCREEN　0，0としてから，

```
OUT＆H4000, 255
```

とすると，8つのドットすべてがセットされ，横線が描かれます。他の色についても同じです。青の場合は，4000Hから7FFFHまですべてに255＝FFHを送ることで，画面全体を塗りつぶせます。これを機械語でプログラムにしたものが〔プログラム9-1〕です。BASICでCLEAR＆HEFFF

〔図9-6〕　〔図9-7〕

と指定してからモニタのMコマンドで打ち込んでください。間違いがないか確認したら，G F000として実行することができます。この機械語プログラムの働きをBASICで置きかえたのが〔プログラム9-2〕です。右側は機械語プログラムをアセンブリ言語で，表現したものです。BASICのプログラムを見れば，どのような動作をしているのかが理解できるでしょう。また，このプログラムもそのままRUNで走りますから，機械語との速度の違いを試してみてください。

〔プログラム9-1〕

```
:F000=01  00  40  21  FF  3F  ED  78  /..ﾛ!ﾃ?▞x
:F008=3E  FF  ED  79  03  2B  7C  B5  />ﾃ▞y.+|ｵ
:F010=C2  08  F0  C9  00  00  00  00  /ﾂ.▒/....
:F000=01  00  40             LD  BC,4000H
:F003=21  FF  3F             LD  HL,3FFFH
:F006=ED  78                 IN  A(C)
:F008=3E  FF                 LD  A,3FFFH
:F00A=ED  79                 OUT(C),A
:F00C=03                     INC  BC
:F00D=2B                     DEC  HL
:F00E=7C                     LD  A,H
:F00F=B5                     OR  L
:F010=C2  08   F0            JP  NZ,F008H
:F013=C9                     RET
```

〔プログラム9-2〕

```
10 WIFTH80:SCREEN0,0:CLS4
20 GOSUB 1000
30 END
40 '
50 '
60 '
70 '
80 '
90 '
1000 BC=&H4000              :F000=10040           LD BC,4000H
1010 HL=&H3FFF              :F003=21 FF 3F        LD HL,3FFFH
1020 A=INP(BC)              :F006=ED 78           IN A(C)
1030 A=&HFF                 :F008=3E FF           LD A,3FFFH
1040 OUT BC, A              :F00A=ED 79           OUT(C),A
1050 BC=BC+1                :F00C=3               INC BC
1060 HL=HL-1                :F00D=2B              DEC HL
1070                        :F00E=7C              LD A,H
1080                        :F00F=B5              OR L
1090 IF HL<>0 GOTO 1030     :F010=C2 8  F0        JP NZ,F008H
1100 RETURN                 :F013=C9              RET
```

また，〔プログラム9-2〕をよく見るとわかりますが，BCに指定するアドレスを8000H，C000Hとすればそれぞれ赤，緑で塗りつぶしを行なうことができます。機械語プログラムを書きかえ

るときに注意しなくてはいけないのは，このアドレスの書き方で，16進4桁の数を扱うには上位8ビットと下位8ビットを逆にするという点です。たとえば，アセンブリ言語で，LD　BC, 4000Hとなっている場合，機械語では01 00 40になります。

〔プログラム9-3〕は，この機械語プログラムをBASICからPOKE文で書きかえ，色を指定した後，サブルーチンとして呼び出して使うものです。この場合，色は重ね合わせになりますから，青で塗ったうえに赤を塗るとマゼンタになります。また，CLSは直前で使った色に対して行なわれます。このCLSを行なう方法は，機械語をどのように書きかえているか考えてみてください。

〔プログラム9-3〕

```
10 CLEAR&HEFFF
20 WIDTH 80:SCREEN 0,0:CLS 4
30 INPUT "CLS = 0 : ｱｵ = 1 : ｱｶ = 2 : ﾐﾄﾞﾘ = 3 ";COL
40 IF COL<0 OR COL>3 GOTO 30
50 ON COL+1 GOSUB 70,100,100,100
60 GOTO 30
70 POKE &HF009,0
80 CALL &HF000
90 RETURN
100 COL=COL*64
110 POKE &HF009,255:POKE &HF002,COL
120 CALL &HF000
130 RETURN
```

●機械語によるソート

データを大きい順や小さい順に並べかえるソートは，データが多いときにはかなりの時間がかかります。今度はこのソートの簡単なものを機械語で作ってみましょう。

〔プログラム9-4〕

```
:F000=21 1C F0                    LD HL,F01CH
:F003=06 FF                       LD B,FFH
:F005=7E                          LD A,(HL)
:F006=48                          LD C,B
:F007=54                          LD D,H
:F008=5D                          LD E,L
:F009=BE                          CP (HL)
:F00A=D2 12 F0                    JP NC,12F0H
:F00D=08                          EX AF,AF'
:F00E=7E                          LD A,(HL)
:F00F=08                          EX AF,AF'
:F010=77                          LD (HL),A
:F011=08                          EX AF,AF'
:F012=23                          INC HL
:F013=0D                          DEC C
:F014=20 F3                       JR NZ,F3H
:F016=EB                          EX DE,HL
:F017=77                          LD (HL),A
:F018=23                          INC HL
:F019=10 EA                       DJNZ EAH
:F01B=C9                          RET
```

〔プログラム9-4〕が機械語によるソートのプログラムです。右側はアセンブリ言語で表現したものです。打ち込むときはこの部分は無視してください。このプログラムでは，ソートするデータ数は255個，扱える数値データは0から255までの整数に限られます。このデータは，F01CHから順に書き込んでおきます。ソートが完了すると，同じアドレスから大きい順に並べかえら

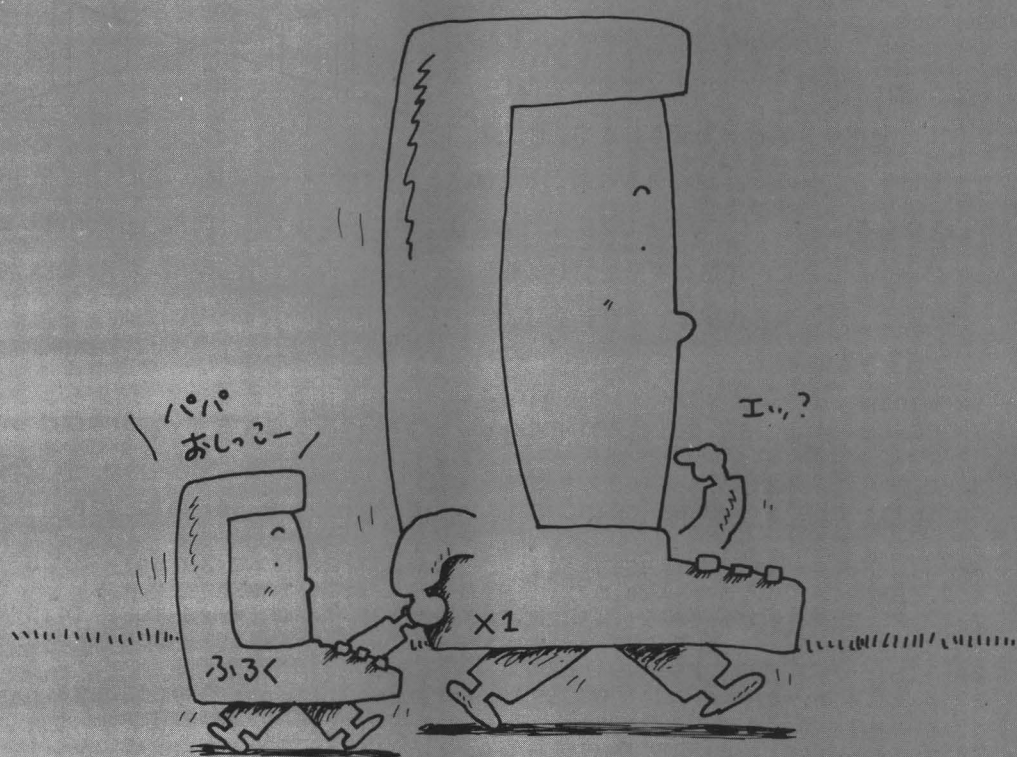
れて入っています。間違いなく打ち込んだらBASICに戻り，〔プログラム 9-5〕を打ち込んでください。

〔プログラム 9-5〕

```
10 CLEAR &HEFFF
20 FOR I=1 TO 255
30 DA=INT(RND*255)+1
40 POKE &HF01B+I,DA
50 PRINT DA,
60 NEXT:PRINT
70 PRINT "ｶｷｺﾐ ｵﾜﾘ":STOP
80 CALL &HF000
90 PRINT "SORT ｵﾜﾘ":STOP
100 FOR I=1 TO 255
110 PRINT PEEK(&HF01B+I),
120 NEXT
```

ここでは，サンプルデータとして，1から255までの数値を乱数を使用して発生させ，画面に表示しながらF01CHから書き込んでいます。書き込みが終わるとSTOP文で止まります。CONTを実行すると，1秒もかからずに「SORT　オワリ」のメッセージがでます。再びCONTを実行すると，F01CHから書き込まれたソートずみのデータを読み出して表示します。

このプログラムでは，データの数をF004Hに書き込んであります。このFFHを変えることにより，ソートするデータの数を決めることができます。たとえば，10個のデータをソートしたいときには，0AHを書き込みます。また，F00AHからを，JP C，F012H，機械語ではDA 12 F0とすれば小さい順のソートになります。

付　録

立体迷路から脱出！

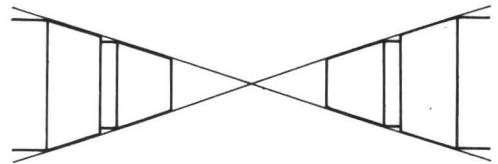
〔３Ｄの技法〕

３Ｄグラフィックという言葉がよく聞かれますが，この３Ｄというのは３Dimensional，すなわち３次元グラフィックのことで，平面（２次元）の画面に立体（３次元）のものを表現しようとするものです。

身近にあるものにたとえると，写真などがよい例です。立体的な建物や風景も写真に撮ると平面の中におさまってしまいます。それでも，写真を見れば建物の立体感がわかりますね。これは，近くのものは大きく，遠くのものは小さくという遠近感が写真の中にあるからです。コンピュータのディスプレイ画面にもこれと同様な操作をしてやれば，立体を表現できるはずですね。これが３Ｄグラフィックです。

３Ｄ－２Ｄ変換の方法としてよく使われるのが，単点（１点）透視と呼ばれる方法です。図10－１は，同じ高さの４つのビルを単点透視で描いたものです。同じ高さの線が奥へいくほど１点に集まってますね。このように３次元，２次元に変換する方法を単点透視といいます。

ここでは，この単点透視を用いて作った迷路脱出ゲームを紹介します。

〔図10－１〕

ゲームのやり方は簡単です。プログラムを走らせると，画面には現在の位置（地図），方向（コンパス），見ている景色が表示されます。前に進むときはテンキーの8，左右後ろに向くときは，それぞれテンキーの4，6，2で操作します。進めるのは前だけで，たとえば左にいきたいときは，4でまず左を向き，8で前進します。３つの画面を参考に，迷路を脱出してください。

簡単にプログラムの説明をしておきましょう（プログラム10－１）。

〔プログラム10－１〕

```
10 WIDTH 40:INIT
20 COLOR 6:LOCATE 5,10:PRINT"data ﾖﾐｺﾐﾁｭｳ":LOCATE 8,12:PRINT"ｼﾊﾞ
ﾗｸ ｵﾏﾁｸﾀﾞｻｲ"
30 CLEAR:DIMm%(66,66):GOSUB 590
40 INIT: CLS 4 :GOSUB 50:SCREEN 1,1:CLS4:GOSUB 50:GOTO 100
50 '
60 LINE(0,0)-(57,58),PSET,1,b:PSET(57,57,2):LINE(63,54)-(59,57)-
(63,60):LINE(59,57)-(70,57):LINE(0,99)-(161,199),PSET,4,b
70 CIRCLE(220,50),40,7:CIRCLE(220,50),35,7:FOR rr=0 TO 3.14159*2
 STEP 3.14159/4:LINE(COS(rr)*40+220,SIN(rr)*40+50)-(COS(rr)*35+2
20,SIN(rr)*35+50),PSET,6:NEXT
80 nx=216:ny=0:mn$="N":GOSUB90:ny=95:nx=217:mn$="S":GOSUB90:nx=1
68:ny=48:mn$="W":GOSUB 90:nx=264:mn$="E":GOSUB 90:RETURN
90 mn=ASC(mn$):mn$=LEFT$(CGPAT$(mn),8):POSITION nx,ny:COLOR7:PAT
TERN-3,mn$:RETURN
100 '
110 m%(1,0)=1:m%(0,1)=1:sc=0
120 X=1:Y=1:PSET(X,Y,6):HB=3:vx=0:vy=0:A=8:GOTO 160
130 '
140 A=STICK(0):IF (A MOD 2)=1 OR A=0 THEN 140
150 IF L1$="000000" THEN INIT:CLS4:CSIZE 3:LOCATE 4,10:PRINT#0"C
ONGRATURATION":END
```

```
160 DI=0:vx=0:vy=0:COLOR0:GOSUB290
170 A=A¥2:ON A GOTO 180,190,200,210
180 A=3:GOTO 220
190 A=4:GOTO 220
200 A=2:GOTO 220
210 A=1:DI=1
220 HB=(((HB+A-2) MOD 4)+1)
230 rr=(HB-2)*90:rr=rr/180*3.1415926:vx=COS(rr):vy=SIN(rr):gx=vx
:gy=vy
240 IF m%(X+VX,Y+VY)=1 THEN VX=0:VY=0
250 COLOR7:GOSUB 290
260 IF DI=0 THEN VX=0:VY=0
270 SCREEN sc,0:PSET(x,y,5):SCREEN sc,1:PSET(x,y,5):x=x+vx:y=Y+v
y:PSET(x,y,6):SCREEN sc,0:PSET(x,y,6):SCREEN sc,sc
280 WINDOW(0,0)-(319,199),(0,0)-(319,199):GOSUB 300:SCREEN sc,sc
:WINDOW(0,0)-(319,199),(0,0)-(319,199):GOTO 130
290 rr=(HB*90-180)*3.1415926/180:SCREEN sc,0:LINE(220,50)-(COS(r
r)*30+220,SIN(rr)*30+50):SCREEN sc,1:LINE(220,50)-(COS(rr)*30+22
0,SIN(rr)*30+50):RETURN
300 '
310 L1$="":L2$="":L3$=""
320 IF HB=1 THEN IF Y<8 THEN JK=Y-1 ELSE JK=6 ELSE JK=6
330 IF HB=4 THEN IF X<8 THEN JK=X-1 ELSE JK=6
340 FOR I=0 TO JK
350 L1$=L1$+RIGHT$(STR$(M%(X+gY+I*gX,Y-gX+I*gY)),1)
360 L2$=L2$+RIGHT$(STR$(M%(X-gY+I*gX,Y+gX+I*gY)),1)
370 L3$=L3$+RIGHT$(STR$(M%(X+gX+I*gX,Y+gY+I*gY)),1)
380 NEXT
390 '
400 SCREEN sc,1-sc:sc=1-sc:WINDOW(1,100)-(160,198),(0,0)-(160,10
0) :CLS 0
410 IF MID$(L1$,1,1)="1" THEN LINE(0,0)-(10,4)-(10,92)-(0,100) E
LSE LINE(0,4)-(10,4)-(10,92)-(0,92)
420 IF MID$(L2$,1,1)="1" THEN LINE(160,0)-(150,4)-(150,92)-(160,
100) ELSE LINE(160,4)-(150,4)-(150,92)-(160,92)
430 IF MID$(L3$,1,1)="1" THEN LINE(10,4)-(150,92),PSET,7,B:RETUR
N
440 IF MID$(L1$,2,1)="1" THEN LINE(10,4)-(40,18)-(40,68)-(10,92)
 ELSE COLOR0:LINE(10,4)-(40,18):LINE(10,92)-(40,68):COLOR7:LINE(
10,18)-(40,18)-(40,68)-(10,68)
450 IF MID$(L2$,2,1)="1" THEN LINE(150,4)-(120,18)-(120,68)-(150
,92) ELSE COLOR0:LINE(150,4)-(120,18):LINE(150,92)-(120,68):COLO
R7:LINE(150,18)-(120,18)-(120,68)-(150,68)
460 IF MID$(L3$,2,1)="1" THEN LINE(40,18)-(120,68),PSET,7,B:RETU
RN
470 IF MID$(L1$,3,1)="1" THEN LINE(40,18)-(55,24)-(55,56)-(40,68
) ELSE  COLOR0:LINE(40,18)-(55,24):LINE(40,68)-(55,56):COLOR7:LI
NE(40,24)-(55,56),PSET,7,B
480 IF MID$(L2$,3,1)="1" THEN LINE(120,18)-(105,24)-(105,56)-(12
0,68) ELSE COLOR0:LINE(120,18)-(105,24):LINE(120,68)-(105,56):CO
LOR7:LINE(120,24)-(105,56),PSET,7,B
490 IF MID$(L3$,3,1)="1" THEN LINE(55,24)-(105,56),PSET,7,B:RETU
RN
500 IF MID$(L1$,4,1)="1" THEN LINE(55,24)-(65,28)-(65,47)-(55,56
) ELSE LINE(55,28)-(65,47),PSET,7,B
510 IF MID$(L2$,4,1)="1" THEN LINE(105,24)-(95,28)-(95,47)-(105,
56) ELSE LINE(105,28)-(95,47),PSET,7,B
520 IF MID$(L3$,4,1)="1" THEN LINE(65,28)-( 95,47),PSET,7,B:RETU
RN
530 IF MID$(L1$,5,1)="1" THEN LINE(65,28)-(70,31)-(70,43)-(65,47
) ELSE LINE(65,31)-(70,43),PSET,7,B
540 IF MID$(L2$,5,1)="1" THEN LINE(95,28)-(90,31)-(90,43)-(95,47
) ELSE LINE(95,31)-(90,43),PSET,7,B
550 IF MID$(L3$,5,1)="1" THEN LINE(70,31)-( 90,43),PSET,7,B:RETU
RN
```

```
560 IF MID$(L1$,6,1)="1" THEN LINE(70,31)-(75,33)-(75,39)-(70,43
) ELSE LINE(70,33)-(75,39),PSET,7,B
570 IF MID$(L2$,6,1)="1" THEN LINE(90,31)-(85,33)-(85,39)-(90,43
) ELSE LINE(90,33)-(85,39),PSET,7,B
580 LINE(75,33)-( 85,39),PSET,7,B:RETURN
590 '
600 RESTORE 680
610 FOR i=0 TO 7:FOR j=0 TO 7
620 READ a$ :a$=HEXCHR$(a$):d$=""
630 FOR l=0 TO 8
640 d$=d$+RIGHT$("0000000000"+BIN$(ASC(MID$(a$,l+1,1))),8)
650 NEXT
660 FOR l=1 TO 64: m%(i*8+j,l-1)=VAL(MID$(d$,l,1)):NEXT
670 NEXT :NEXT:RETURN
680 DATABFFFFFFFFFFFFFC0,A000800882020C0,AFFEBBEFABEBEEC0,8882A
220220808C0,BAAEAEBFFEFFFBC0,8A2882A0000820C0,AFEBFEAFFFFAEFC0,A
02A00880220220C0
690 DATABBEAFFFBFAFFAEC0,8A0A02020A82A2C0,AAFBFEFEBAFABAC0,A8088
080A22282C0,AFFEBFBFEFAEFEC0,A0828888820A2C0,AFBAAAEABBEFABC0
700 DATAA022222A0800A8C0,BFEFFEEBEBFEAEC0,820202828022AC0,BAFEE
FAFAFEBEAC0,8A888828220A28C0,EEEBFBEBFEFAEFC0,A28A0A08200888C0,B
ABABAFFAFEFBEC0
710 DATAA2028A88822022C0,AFFEEEAAFEBFEAC0,A80888A2208008C0,AAEBF
BBEEFBFFFC0,AA22002080A008C0,ABBEFFFFBFAFBAC0,8882200020202082C0,B
EEEEFFFEFFEFEC0
720 DATA8028800822088 0C0,BFEBBFEAFABBBFC0,8028808A08A220C0,BFAEF
EBBEFAEFEC0,822082A2208800C0,BAFFBEAEBAFBFFC0,8A0008A88A0800C0,B
BBFFBABBBBFFEC0
730 DATAA22000888A2082C0,EEEFFEFBEBEEBEC0,880002882A0A88C0,BFFFB
BAFEAFABBC0,800088222222280C0,BEFEEFFABFAEFFC0,828A200880A200C0,B
AABBEBFFEAAFEC0
740 DATA8A2820A020AA88C0,ABEFFFEFAFAEAFC0,A8200020A028A0C0,BBFFF
BBEBFEBFEC0,A2002082A22808C0,AEFFEFFAAAAFABC0,A2008008AA82A8C0,B
BEFBFEFAAFABEC0
750 DATAA228228A2A0A80C0,AEABFABAEAEAFFC0,A0A0020808880000,FFFFF
FFFFFFFFFC0,FFFFFFFFFFFFFFC0,0000000000000000,0000000000000000,0
000000000000000,0000000000000000
```

10行から30行で迷路のデータを配列m%に読み込んでいます。実際の読み込みは590行以降のサブルーチンで行なっています。迷路をつくるプログラムはむずかしく，プログラム自体長くなってしまうので，ここではすでに作られた迷路をデータにしています。時間のある方は，自分で迷路を作るプログラムをつけ加えてください。

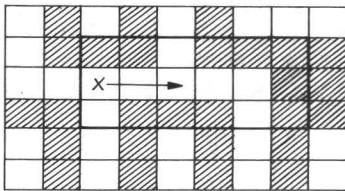
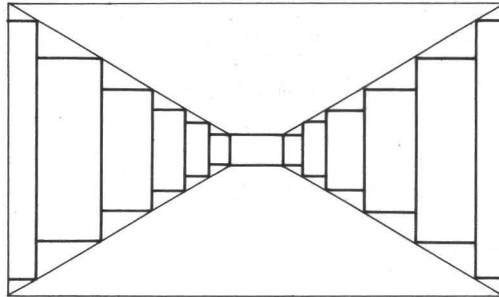
迷路は64×64ドットで作られ，道があれば0，なければ（壁なら）1として，64×64ドットの2進数データを作ります。これを横1列ずつとって16進データに直し，8バイト×64個のデータとしてDATA文に書いておきます。

40行ではゲーム画面を作っています。そのサブルーチンが50行から90行です。このゲームでは，SCREENモードをかえて，画面を切りかえているので，2枚分のゲーム画面を用意します。

110行から120行が初期設定になります。変数SCには現在表示中のSCREENモード（0あるいは1）が入ります。SCREEN SCを表示中，裏のSCREEN1-SCに描き込み，描き込みが終わると同時に画面を切りかえます。こうすることで，画面を瞬時に変えることができます。変数X，Yが現在いる座標，HBが向いている方向，VX，VYは進む方向をベクトルで表わしたものです。方向は北を1とし，右まわりに2，3，4とします。

メインルーチンは130行から290行になります。ここでは，テンキーで入力された値から進む方向を求め，画面の書きかえをします。270行では地図の書きかえ，280行では見ている景色の

3D画面を描きます。これらの処理が終わると130行（メインルーチンの頭）に戻り，同じ作業を繰り返します。

300行からはじまるサブルーチンは，3D画面作成のための情報をm%から引き出してくる，という作業をします。この情報というのは，図10－2に示すように現在地を基点とする3×7の点です。これらの点の情報が，左から，L1$，L3$，L2$と列ごとに格納されます。これらの点を順に調べて迷路の図を描いていくわけです。この作業は400行から580行のサブルーチンで行なわれます。図10－3は基本となる画面で，壁に沿って道がなければ実線で，道があれば破線のように描けば良いわけです。

〔図10－2〕　〔図10－3〕

このプログラムでは一番奥の点（正面7つ目のブロック）を壁にしていますから，遠くが突き当たりのように見えても実際にそうだとは限りません。また，画面の情報として6ブロックしか使っていませんが，7ブロック，8ブロックと情報を多くして，画面の奥行きを広くすることもできます。挑戦してみてください。

キャラクタ・ジェネレータ

次のプログラムは，カーソルとテンキーでキャラクタを描き，そのキャラクタのデータをプログラムに収めるものです。ゲームなどで使うキャラクタを簡単に作ることができます。

〔プログラム10－2〕

```
10 CLEAR:DIMP%(7,7),B$(3):INIT:WIDTH40:CLS4
20 LOCATE 0,12:PRINT"ｹﾞﾝｻﾞｲﾉ DATAｦ CLEAR ｼﾏｽｶ ? Yes/No"
30 A$=INKEY$:IFA$=""THEN30ELSEIFA$="N"ORA$="n"THENCLS:GOTO50ELSE
IFA$="Y"ORA$="y"THEN40ELSEGOTO30
40 CLS:A$=STRING$(24,CHR$(0)):FORI=0TO255:DEFCHR$(I)=A$:NEXT
50 COLOR1:FORI=0TO16:LOCATEI,0:PRINT"■";:LOCATE17-I,17:PRINT"■";
:LOCATE0,17-I:PRINT"■";:LOCATE17,I:PRINT"■";:NEXT
60 FORI=0TO15:LOCATE0,I+1:PRINTHEX$(I);:LOCATEI+1,0:PRINTHEX$(I)
;:NEXT:COLOR7
70 CGEN1:FORI=0TO15:FORJ=0TO15:LOCATEI+1,J+1:PRINT#0CHR$(J+I*16)
;:NEXTJ,I:CGEN
80 COLOR6:FORI=0TO9:LOCATEI+25,13:PRINT"*";:LOCATE34-I,23:PRINT"
*";:LOCATE34,14+I:PRINT"*";:LOCATE25,23-I:PRINT"*";:NEXT
90 FORI=0TO7:IFI=0THENCOLOR7:CREV1ELSECOLORI
100 LOCATE26+I,12:PRINTHEX$(I);:LOCATE24,14+I:PRINTHEX$(I);:CREV
:NEXT:COLOR7
110 LOCATE1,1:CFLASH1:CGEN1:COLOR7:PRINT#0CHR$(0);:CGEN:CFLASH
120  LOCATE19,1:COLOR5:PRINT#0CHR$(30,31,29,28);:COLOR7:PRINT" ﾁ
ｲｻｲ ｶﾞﾒﾝﾉ ｶｰｿﾙ "
130  LOCATE19,3:COLOR5:PRINT"CLR";:COLOR7:PRINT;"  ﾁｲｻｲ ｶﾞﾒﾝﾉ ｸﾘ
ｱ"
140  LOCATE19,5:COLOR5: PRINT " i"
150  LOCATE19,6:COLOR5:PRINT"j";:COLOR7:PRINT"+";:COLOR5:PRINT"K
";:COLOR7:PRINT "  ｵｵｷｲ ｶﾞﾒﾝﾉ ｶｰｿﾙ"
160  LOCATE19,7:COLOR5:PRINT" m"
170  LOCATE19,9:COLOR5:PRINT" c";:COLOR7:PRINT"   ﾃﾞｰﾀ ｺﾋﾟｰ"
180  LOCATE0,24:COLOR5:PRINT"     o";:COLOR7:PRINT"  DATA OUT";
190 LOCATE1,1:CFLASH1:CGEN1:COLOR7:PRINT#0CHR$(0);:CGEN:CFLASH
200 LABEL"MAIN"
210 CX=1:CY=1:X=1:Y=1:MN$="01234567ijmKco"+CHR$(28,29,30,31,12,1
3)
220 LABEL"LOOP"
230 LOCATECX+25,CY+13:COLORP%(CX-1,CY-1):CFLASH1:PRINT"●";:CFLAS
H:CH=X*16+Y-17
240 Q$=INKEY$:IFQ$=""THEN240ELSEW=INSTR(MN$,Q$):IFW=0THEN240
250 IFW<9THEN"PSET"ELSEIFW<13THEN"CURS"ELSEIFW=13THEN"COPY"ELSEI
FW=14THEN"DATAOUT"ELSEIFW<19THEN"LOCA"ELSEIFW=19THEN"CLR"ELSEGOT
O"FINE"
260 LABEL"PSET"
270 P%(CX-1,CY-1)=W-1
280 LOCATECX+25,CY+13:COLORW-1:PRINT"●";
290 CX=CX+1:IFCX>8THENCX=1:CY=CY+1:IFCY>8THENCX=8:CY=8
300 GOTO"LOOP"
310 LABEL"LOCA"
320 LOCATECX+25,CY+13:COLORP%(CX-1,CY-1):PRINT"●";:ONW-14GOSUB33
0,350,370,390:GOTO"LOOP"
330 CX=CX+1:IFCX>8THENCX=1:CY=CY+1:IFCY>8THENCX=8:CY=8
340 RETURN
350 CX=CX-1:IFCX<1THENCX=8:CY=CY-1:IFCY<1THENCY=1:CX=1
360 RETURN
370 CY=CY-1:IFCY<1THENCY=1
380 RETURN
390 CY=CY+1:IFCY>8THENCY=8
```

```
400 RETURN
410 LABEL"FINE"
420 CFLASH0:LOCATECX+25,CY+13:COLORP%(CX-1,CY-1):PRINT"●";:CH=X*
16+Y-17:GOSUB"DEF-CHR":GOTO"LOOP"
430 LABEL"CURS"
440 CONSOLE0,25,0,40:CGEN1:COLOR7:LOCATEX,Y:CFLASH:PRINT#0CHR$(X
*16+Y-17);
450 ONW-8GOSUB460,480,500,520:GOTO540
460 Y=Y-1:IFY<1THENY=1
470 RETURN
480 X=X-1:IFX<1THENX=1
490 RETURN
500 Y=Y+1:IFY>16THENY=16
510 RETURN
520 X=X+1:IFX>16THENX=16
530 RETURN
540 CFLASH1:COLOR7:LOCATEX,Y:PRINT#0CHR$(X*16+Y-17);:CGEN:CFLASH
:CONSOLE14,9,26,8
550 GOTO"LOOP"
560 LABEL"CLR"
570 CONSOLE14,9,26,8:FORI=0TO7:FORJ=0TO7:P%(I,J)=0:NEXTJ,I:PRINT
CHR$(12):GOTO"LOOP"
1000 LABEL"DEF-CHR"
1010 A$="":FORI=0TO7:FORJ=0TO7:A$=A$+RIGHT$("000"+BIN$(P%(j,i)),
3):NEXTJ,I
1020 B$(1)="":B$(2)="":B$(3)=""
1030 FORI=0TO7:FORJ=0TO7:FORL=0TO2
1040 B$(L+1)=B$(L+1)+MID$(A$,L+J*3+I*24+1,1)
1050 NEXTL,J,I
1060 FORI=1TO3:L$="":FORJ=0TO7:L$=L$+RIGHT$("00"+HEX$(VAL("&B"+M
ID$(B$(I),J*8+1,8))),2):NEXT:B$(I)=L$:NEXT
1070 DEFCHR$(CH)=HEXCHR$(B$(3)+B$(2)+B$(1))
1080 COLOR7:CONSOLE0,25,0,40:FORi=1TO3:LOCATE0,17+i:PRINTb$(4-i)
;:NEXT:CONSOLE14,9,26,8
1090 CX=1:CY=1:RETURN
1100 LABEL"COPY"
1110 A$=RIGHT$(CGPAT$(CH),24)
1120 FORJ=0TO2:B$(J+1)="":FORI=0TO7:B$(J+1)=B$(J+1)+RIGHT$("0000
0000"+BIN$(ASC(MID$(A$,I+J*8+1))),8):NEXTI,J
1130 FORI=0TO7:FORJ=0TO7:P%(J,I)=VAL("&b"+MID$(B$(3),J+I*8+1,1)+
MID$(B$(2),J+I*8+1,1)+MID$(B$(1),J+I*8+1,1)):COLORP%(J,I):LOCATE
J+26,I+14:PRINT"●";:NEXTJ,I:COLOR7
1140 GOTO"LOOP"
1200 LABEL"DATAOUT"
1210 WIDTH40:CLS:COLOR4,0:LINE(0,0)-(17,17),"■",B:COLOR7:LOCATE0
,20:PRINT"DATA ｼﾞﾄﾞｳ ｻｸｾｲ !!!":GYO=2000:I=0
1220 IFRIGHT$(CGPAT$(I),24)=STRING$(24,CHR$(0))THEN"INC"
1230 SCREEN0,0:CGEN1:LOCATEINT(I/16)+1,(I MOD 16)+1:PRINT#0CHR$(
I):CGEN
1240 SCREEN0,1:GYO$=RIGHT$(STR$(GYO),4):DD$="":C$=RIGHT$(CGPAT$(
I),24)
1250 FORj=0TO23:DD$=DD$+RIGHT$("00"+HEX$(ASC(MID$(C$,j+1,1))),2)
:NEXT
1260 CLS:PRINTGYO$+"DA."+DD$+","+HEX$(I)
1270 KEY0,CHR$(30,30)+CHR$(13)+"GOTO1280"+CHR$(13):END
1280 GYO=GYO+10
1290 LABEL"INC":I=I+1:IF I<256 THEN 1220
1300 SCREEN 0,1:CLS:SCREEN0,0:CLS:LOCATE0,10:PRINT"DATAﾊ 2000ｶﾗ
ﾃﾞｽ":END
```

プログラムの説明を簡単に行ないましょう。まず，10行から40行で，画面設定，配列宣言などの初期設定を行ないます。配列P%は作成中のキャラクタの情報を格納しておくもので，B$には色別データが入ります。40行では，255個のキャラクタに空のデータを定義します。すでに定

義ずみのキャラクタは未定義状態となります。後で作ったキャラクタのデータがここに書き込まれるわけです。

50行から190行では，キャラクタを作成するときの画面（作成画面）を作っています。画面は2枚あって，1つは1文字分（8×8ドット）のキャラクタ作成用の小さい作成画面，もう1つはそれぞれで作ったキャラクタを，任意のコードに定義するための大きい表示画面です。

200行から250行がこのプログラムのメインルーチンで，2枚の画面にカーソルを表示し，キー入力待ちになります。240行で入力に対する処理の飛び先を指定しています。

ラベル「PSET」ではじまる200行から300行のルーチンでは，キャンバスにキャラクタを描いていく際の点を打つ作業をし，カーソル位置やカラーの判断を行ないます。作成中のキャラクタ（枠の中の情報）は，配列P%に格納されます。

310行から400行は，作成画面のカーソル移動を行なうサブルーチンです。上下左右はもちろん，右端にきたら1段下の左端に移動する，という処理も行なっています。

410行，420行では，キャラクタ定義をするキー(リターンキー)が押されたときの処理をするもので，実際のキャラクタ定義はラベル“DEF-CHR”のサブルーチンで行なわれます。

430行から550行は，大きい画面の方のカーソル移動を行なうサブルーチンです。先に述べましたように，この画面のカーソルを動かすことで任意のコードに，作成画面で作ったキャラクタを定義することができます。

560行，570行ではキャンバスをクリアし，同時に配列P%もクリア（初期化）します。

1000行から1090行が実際のキャラクタ定義ルーチンです。ここでの作業を簡単にまとめると，配列P%に格納された8×8ドット計64個のデータ（0～8のパレットコード）を，2進数3桁のデータに変え192個の文字列を作ります。次にこの文字列を各色（赤，緑，青）のデータごとに分け，配列B$に格納します。この作業は1030行から1050行で行なっています。配列B$のデータは16進に変換され，最後にHEXCHR$で定義されます。

1100行から1140行の「COPY」ルーチンでは，“DEF-CHR”と逆の作業をしています。つまり，定義ずみのキャラクタを配列P%に取り込みキャンバスに表示させます。

1200行から1300行では，作成したキャラクタのデータをDATA文としてこのプログラムに書き加えます。DATA文は2000行以降に書かれます。

以上がプログラムの大まかな説明で，細かい部分についてはあなたの研究課題としてください。

次にこのプログラムの使用方法を説明しましょう。プログラムを走らせると，現在のデータをクリアするかどうかを聞いてきます。YESならy，NOならnを押してください。yが押されると，これまでに定義したキャラクタをクリアして初期状態に戻します。

画面は変わって，大きな枠と小さな枠が表示され，2つの画面にはカーソルが点滅します。小さい方はキャラクタ作成用のキャンバスで，大きい方は作成したキャラクタを表示し，定義する画面です。

キャンバスは，1文字分のキャラクタ（8×8ドット）を拡大して見ていることになります。カーソルキーを使ってカーソルを移動し，0～7の数字をキーインします。するとカーソルのある位置に，数字に対応する色の点が表示されます。同様にして，好きな色で好きなキャラク

タを作ることができます。キャンバスにキャラクタを描き終わったら，大きい画面のカーソルをキャラクタを定義するコードの位置に移動させます。表示画面のカーソルはアルファベットキーを使って動かしますが，このとき必ず小文字 (CAPSLOCKキーを解除する) にしてください。キャラクタコードは16進２桁で，画面の上に表示している数字が上位の桁，左が下位の桁です。

リターンキーを押すと，画面にキャンバスで作ったキャラクタが１文字分の大きさで表示され，同時にキャラクタ定義も行なわれます。画面下には，そのキャラクタのデータが表示されます。

同じキャラクタをいくつも定義したり，定義ずみのキャラクタを修正したりするときは，大きい画面のカーソルをそのキャラクタの位置に移動させ，[C] キーを押します。すると，画面のキャラクタがキャンバスに転送されますから，そこで修正，再定義を行なってください。

キャンバスをクリアするときは [SHIFT] + [CLS HOME] で行ないます。また，定義ずみのキャラクタを取り消すときは，空のキャラクタ（作成画面をクリアしたもの）をそのコードへ定義することで行ないます。

最後に [0] キーを押すと，このプログラムの2000行以降に定義したキャラクタのデータがDATA文として書き込まれます。これをカセットなどにセーブすることで，キャラクタデータを保存することができます。

CAIってなんだ?

パソコンの持つたくさんの可能性の中でも最近は，教育や学習に使えるということが注目されてきています。この項では教育とコンピュータの可能性をさぐりながら，学習プログラム作りにも挑戦していきましょう。

CAI——ちょっと聞きなれない言葉です。CAIとはComputer Assisted Instructionの略で，つまり「コンピュータを用いた教育システム」のことなのです。

この言葉そのものがまだあまり知られていないように,日本でCAIはあまり普及していません。しかし,CAIがまったく行なわれていないというわけではありません。中学校や大学などの教育の現場でも実験的に行なわれつつあります。CAIの先進国は,やはりコンピュータの故郷アメリカです。ニューヨーク州などでは，小，中学校のすべての生徒に1人1台のコンピュータを与えるという目標を立てています。パソコンの急速な普及にともない，日本でも発展が期待される分野といえます。

コンピュータを用いた学習では，同じことを何度でも繰り返して学べるという特徴があります。また，学習者が自分のペースで学習を進められるという特徴もあります。パソコンが先生であれば，出題も採点も自動的にやってくれます。ゲームを取り入れて楽しみながら勉強できるプログラムを組むと，もっと良いでしょう。

それでは，ゲーム形式でちょっと楽しい英語の学習プログラム作りに挑戦してみましょう。

〔プログラム10-3〕

```
10 INIT:CLS 4:WIDTH40:GOSUB 60000
20 LOCATE 5,10:PRINT"1      ｴｲﾀﾝｺﾞ   >>>    ﾆﾎﾝｺﾞ":LOCATE 5,12:PRIN
T"2      ﾆﾎﾝｺﾞ    >>>    ｴｲﾀﾝｺﾞ":LOCATE 0,20:PRINT"———< Push  1-2
 Key  To   Start  >————";
30 A$=INKEY$:IF A$="1" OR A$="2" THEN Q=VAL(A$):CLS:GOTO 40 ELSE
 GOTO 30
40 RESTORE 20000
50 B$="┌"+STRING$(38,"─")+"┐"+"  "+STRING$(38," ")+"|"+"└"+STRING
$(38,"─")+"┘"
60 IF A=1 THEN C$="        ﾂｷﾞﾉ ｴｲｺﾞｦ ﾆﾎﾝｺﾞﾆ ﾅｵｼﾅｻｲ        " ELSE
C$="         ﾂｷﾞﾉ ﾆﾎﾝｺﾞｦ ｴｲｺﾞﾆ ﾅｵｼﾅｻｲ         "
70 B$=LEFT$(B$,41)+C$+RIGHT$(B$,41)
80 CLS:PRINTB$:LOCATE 0,18:PRINTSTRING$(40,"─"):CONSOLE 3,15:KK=
1
100 '
110 CLS:READ QE$,AN$:IF Q=2 THEN SWAP QE$,AN$
115 IF QE$="END" THEN END
120 LOCATE 5,5:PRINTQE$
130 LOCATE 10,8:INPUT NA$
140 IF AN$=NA$ THEN TR=1 ELSE TR=0
150 GOSUB 10000
160 IF TR=1 THEN TT$="ｾｲｶｲ" ELSE TT$="ﾏﾁｶﾞｲ"
170 LOCATE 24,14:PRINTTT$;:PAUSE 30:CLS3:INIT:CONSOLE 3,15
180 IF TR=1 THEN 100
190 IF Q=2 THEN SWAP QE$,AN$
200 CONSOLE 0,25:  J=(KK-1) MOD 5:I=(KK-J-1)¥5:LOCATE I*20,J+19:
PRINTSTRING$(20," ");
```

```
210 LOCATEI*20,J+19:PRINTQE$;:KK=KK+1:IF KK>10 THEN KK=1
220 CONSOLE 3,15:GOTO 100
9999 END
10000 '
10010 CGEN 1:LOCATE 5,15:PRINT#0CHR$(17,33,49)
10020         LOCATE 4,16:PRINT#0CHR$(0,18,34,50)
10030         LOCATE 5,17:PRINT#0CHR$(19,35,51);
10040 GOTO 10070
10050 SOUND6,5:SOUND7,247:SOUND8,14:SOUND11,0:SOUND12,10:SOUND13
,0 :RETURN
10060 FOR I=0 TO  20:NEXT :GOSUB 10050:FOR I=0 TO  35:NEXT:SOUND
 8,0: RETURN
10070 PAUSE 20
10080 FOR L=0 TO 4
10090 FOR J=0 TO 24:A=(J MOD 2)+1+L*2:LOCATE 4,16:PRINT#0CHR$(A)
:GOSUB 10060:NEXT
10100 NEXT
10110  LOCATE 4,16:PRINT#0CHR$(12)
10120 GOSUB 10210
10130 LOCATE 7,14:PRINT#0CHR$(11):PAUSE 3:LOCATE 7,14:PRINTCHR$(
32)
10140 LOCATE 8,13:PRINT#0CHR$(11):PAUSE 3:LOCATE 8,13:PRINTCHR$(
32)
10150 FOR I=9 TO 20:LOCATE I,13:PRINT#0CHR$(11):PAUSE 3:LOCATE I
,13:PRINTCHR$(32):NEXT
10160 LOCATE 21,13:PRINT#0CHR$(11)
10170 FOR I=0 TO 3:LOCATE 21+I,13+I:PRINT#0CHR$(11);:PAUSE 2:LOC
ATE 21+I,13+I:PRINTCHR$(32);:NEXT
10180 LOCATE 25,17:PRINT#0CHR$(11);:POSITION 187,115:COLOR 2:PRW
 4:PATTERN -32,FI$:GOSUB 10210:FOR I=0 TO 16:A=(I MOD 2)+1:PALET
 2,A:PAUSE 3:NEXT
10190 SCREEN 0,0,3:IF TR=1 THEN CIRCLE(201,130),12,3 ELSE LINE(1
90,120)-(210,140),PSET,3:LINE(210,120)-(190,140),PSET,3
10200 LOCATE 25,17:PRINT" ";:CLS1:CLS2:CGEN 0:COLOR7:  RETURN
10210 SOUND6,31:SOUND7,246:SOUND0,255:SOUND1,15:SOUND8,16:SOUND1
1,255:SOUND12,255:SOUND13,0:RETURN
20000 DATAAPPLE,ﾘﾝｺﾞ,BOOK,ﾎﾝ,BOX,ﾊｺ,COFFEE,ｺｰﾋｰ,DESK,ﾂｸｴ,FISH,ｻｶ
ﾅ,HIGH,ﾀｶｲ,LONG,ﾅｶﾞｲ,MORNING,ｱｻ,MUSIC,ｵﾝｶﾞｸ,PARK,ｺｳｴﾝ,ROOM,ﾍﾔ,SN
OW,ﾕｷ,SUMMER,ﾅﾂ,WALL,ｶﾍﾞ,END,END
60000 RESTORE60020:FOR I=0 TO 11:READ A$:DEFCHR$(I)=HEXCHR$(A$):
NEXT
60010 FORI=1TO3:FORJ=1TO3:READA$:DEFCHR$(I*16+J)=HEXCHR$(A$):NEX
TJ,I
60011 READ H$:FI$=HEXCHR$(H$):READ H$:FI$=FI$+HEXCHR$(H$):RETURN
60020 DATA 00606060783F07000060606060783F0700006060783F0700,0000
60607833F0700F0F0F060783F0700A050E060783F0700,00003030383F0700787
87830383F07005028703038 3F0700
60030 DATA 0000060783F070000F0F0783F070000A050E0783F0700,0000
0030383F07000078787 8383F07000050287 0383F0700,00000000081F0F00208
070F0F87F4F00208050A0583F4F00
60040 DATA 000000001F1F002C8078F8787F3FC0248050A8503F1F80,0000
00000007070000 2214C03C3F3FDC00220080142F1788,00000000000070300005
02018FC3F7FB400500008942F57A0
60050 DATA 000000000000101000002120807 6F0F3700021000054B0522,0000
0000001010000042410670F6F0700042000450B4502,0000000000000000000
030606060000 03C7EFFFFFFFF7E3C
60060 DATA 000000000103070F0000000000000000000000000103070F,1F1F
3F7F7FFFFFFFF0000000000000000001F1F3F7E7CFCF8F8,FF7F7F7F7FFFFFFFF081
02040408080AAF870604040 8080AA
60070 DATA 061F3F7FFFFFFFFFF00000000000000000000061F3F7FFFFFFFFFF,FFFF
FF9F1F0F0F9F00000060E0F0F060FFFF0F0707030301,FFFFFFFFFFFFFFFF020
1010000000000AA0301010000000000AA
60080 DATA 0080E0F8FCFCFCFC0000000000000000000080E0F8FCFCFCFC,F8F0
F0E0E0C08080000000000000000000F8F0F0E0E0C08080,00000008080C0C0C0000
00080804040800000000808040408 0
```

```
63000 DATA0000000000080C0F0F0F0F0F0F0F0F0F07070787E37B7F3F3F1F0F
07030000000101030307070F0F9FDFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFF3F00
63010 DATA0080C0C0E0F1FBFBFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFE00206060E0E0E1E3E7EFFFFFEFEFEFEFEFEFCFDFFFFFFFEFEFEFEFEFC
F0E0800000
```

では，プログラムの説明をしましょう。まずフローチャート（図10－4）を見てください。

〔図10－4〕

CAIを実現するためのプログラムの基本的な構造が示されています。フローチャートに合わせてリストを解説していきましょう。

初期設定は10行および60000行からはじまるサブルーチンで，ここでは，画面設定およびキャラクタ定義を行なっています。

続く20行，30行で「出題形式の選択」をします。このプログラムでは，英語から日本語，日本語から英語の２とおりの学習方法があります。

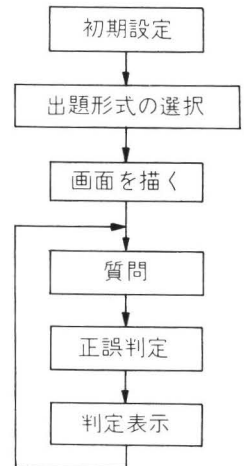
次に，画面を描いているのが50行から80行です。先に選択した学習方法に従って問題文を表示します。

100行から220行がメインルーチンです。110行で問題の読み込み(20000行のDATA文より)，120行で問題表示，130行で解答の入力，140行で正誤の判定をしています。140行のTRは，解答が正解か不正解かを示す変数です。150行で，初期設定で定義されたキャラクタを表示するサブルーチン（10000行）へ飛びます。160行から170行で「セイカイ」，「マチガイ」のメッセージを表示し，不正解の場合は，190行から220行で画面下に誤った単語を表示します。このとき，単語を順序よく並べるためにKKという変数を使って，不正解の数を与えています。再び100行に戻って，メインルーチンが繰り返されます。

このプログラムで使われているキャラクタは，大砲に砲弾，導火線に爆煙の４キャラクタで，コード０～10に導火線，11に砲弾，17～19，33～35，49～51の９つのコードに大砲のキャラクタが定義されます。また，爆煙は32×32ドットの図形で32×32÷８＝128バイトのデータとして変数FI$に格納され，画面にはPATTERN命令で表示されます。これらのキャラクタ定義は60000行からはじまるサブルーチンで行なわれ，キャラクタデータは60020行以降に書かれています。

キャラクタの表示は10000行からはじまるサブルーチンで行なわれ，同時に効果音も出しています。ここで注意したいのが，PRINT＃０という命令です。例えば，PRINT＃０ CHR$(12)と書くと，コード12に定義されたキャラクタを表示することができます。ところがPRINT CHR$(12)とすると，画面をクリアする命令ですね。PCGで定義したキャラクタを表示するときは，この点に気をつけてプログラムしてください。

細かい部分の説明は省略しました。後はプログラムを実行しながら確めてください。

ここに取り上げたプログラムは，CAIの１つの例にしかすぎません。最初にあげたフローチャートを手がかりに，あなた独自のプログラムを作ってみるのもいいでしょう。まずは，参考までにプログラムを走らせてください。

簡易ワードプロセッサ

X1で扱える文字キャラクタには，英数字，カナ，グラフィック文字(図形)などがありますが，漢字ROM（オプション）を本体内に取りつけることで漢字を扱うこともできるようになります。

そこで，この漢字キャラクタを使って簡単な日本語ワードプロセッサ，名付けて「簡易ワードプロセッサ」を作ってみました。

ワードプロセッサ(略してワープロ)とは，漢字，かな混じりの文書を「作成」，「訂正」，「編集」，「印刷」，「保存」するものですが，X1でもプログラム次第でワープロに早変わりします。ただし，簡易型ということなので，専用機のカナ漢字変換などの機能はなく，漢字の入力はJIS漢字コードで入力します。また，「訂正，編集」についてはBASIC同様，カーソルコントロールキー，DELETEキー，CLRキーを使ったスクリーンエディタで行ないます。

●使用法

プログラムのロードが終わったら，文書保存用の生テープ，あるいはディスケットをセットした後にRUNさせます。

数秒後，ブルーの文書作成画面が表示されます。作成画面は，1画面1ページとして3ページ用意されています。現在表示中のページ数は画面左下に表示されます。

1ページの構成は，横37文字×縦10行で，カーソルキーを使って任意の位置に文字を書くことができます。各ページは画面では別ですが実際には上下につながっていると考えてください。カーソルが表示ページの右下にきたら，次のページ（3ページの場合は1ページ）の先頭に移動します。

<各キーの説明>

○[A]～[Z]キー

小文字のa～z。[CAPSLOCK]または[SHIFT]キーを使うと大文字のA～Zが表示されます。

○[カナ]キー

ひらがなを表示します。[CTRL]＋[X]キーでカタカナ表示に切りかえることができます。

○[GRAPH]キー

プログラム中で設定された漢字を表示します。プログラムのDATA文を変更することで，文字を変えることもできます。[CTRL]＋[X]キーで各種記号表示に切りかわります。

○[DEL]キー，カーソルコントロールキー

BASICの場合と同様な働きをします。

○[CLR]・[HOME]キー

これらは各ページごとに働きます。[CLR]キーについては，安全のため“Are You Sure(Y/N)”のメッセージが出ますので，YかNで答えてください。Yの場合，表示中のページのみクリ

アされます。

○ CTRL キー

以下のキーと合わせて使います。

+H キー　DELと同じ

+K キー　HOMEと同じ

+L キー　CLRと同じ

+P キー　ページ1～3をプリンタに出力

+Q キー　JISコード→漢字変換

"CODE"を表示し入力を求めてきますから，表示したい漢字のJISコードを入力すると，カーソル位置にJISコードに対応する漢字を表示します。

+T キー　3ページ分の文書をテープ，またはディスケットにSAVEする。

"FILE NAME"と表示されるので任意のファイル名をつけてください。

+W キー　文書をテープ，またはディスケットからLOADする。

SAVEした文書のファイル名を指定します。

〈プログラムの説明〉

このプログラムには重要なポイントが2つあります。その1つである漢字（ひらがな，カタカナ）出力について説明しましょう。

キーボードから入力された文字（コード）は180行でコントロールコードかどうかを判定されます。コントロールコードでなければ，JISコードとして変数Aに格納されます。

このJISコードを文字に変換して表示しているのが300行です。これ以前にも，ひらがなやカタカナの場合の操作や，行単位で配列に格納するなどの処理が行なわれていますが，ここでは説明を省略します。

さて，300行を見てください。JISコードから文字を得るにはKANJI$関数というものを使います。KANJI$(A)でJISコードAに対応する漢字のパターン(縦16×横16ドット)のコードを得ます。これは，ドットパターン（8×32ドット）でしかないので，POSITION，PATTERN文と共に使わなければなりません。PATTERN文はコード化されたドットパターン(関数KANJI$で得たコード)をPOSITIONで指定した位置に表示するステートメントです。これについて説明しましょう。

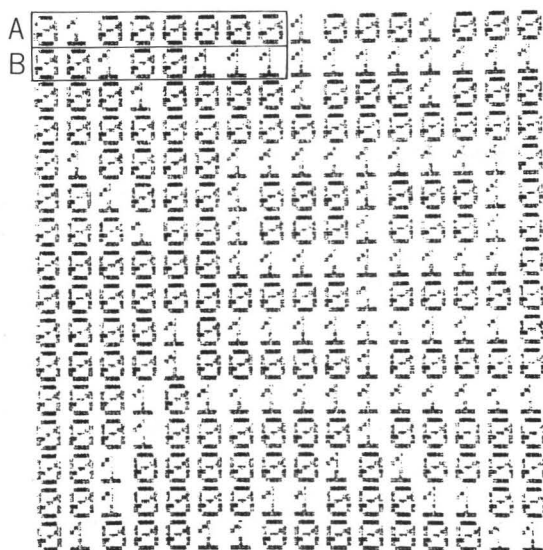
```
10 CLS:A$=KANJI$(&H7F1):PATTERN -16,A$
20 FOR I=0 TO 31
30 LOCATE (I ¥ 16)*8,I MOD 16+3
40 PRINTRIGHT$("0000000"+BIN$(ASC(MID$(A$,I+1,1))),8)
50 NEXT:INIT
60 HCOPY 4
```

KANJI$は，32文字（コントロールコードを含むこともある）の文字列からなり，1文字が，横8bit分の点を表わします。例えばKANJI$(&H7F1)の第1文字は@で，これのASCIIコードは&H40です。これを2進数に直すと$01000000_{(2)}$となり，図10－5のAの部分の点を表わします(図10－6はKANJI$(&H7F1)の文字をプリントしたものです)。

```
300 POSITION X*17+4,Y*17+6:PATTERN -16,KANJI$(A):X=X+1:IF X>36 T
HEN X=0:Y=Y+1:IF Y>9 THEN P=P+1:GOSUB "PAGE":X=0:Y=0
310 RETURN
320 LABEL "CURSOR"
330 J=X*17+4:K=Y*17+22:LINE(J,K)-(J+16,K),PSET,P*1.4:LINE(J,K)-(
J+16,K),PRESET,PP:RETURN
340 LABEL "KEY IN"
350 B$(0)=INKEY$:B$(1)=RIGHT$("0000000"+BIN$(ASC(INKEY$(2))),8)
360 IF MID$(B$(1),2,1)="1" THEN F=0:RETURN ELSE F=-1:RETURN
370 LABEL "PAGE":IF P>3 THEN P=1 ELSE IF P<1 THEN P=3
380 PP=P:INIT:COLOR 1,1:LINE(0,0)-(79,24),CHR$(&H87),BF:COLOR 7:
LOCATE 0,24:PRINT P;
390 FOR I=0 TO 37:J=I*17+4:IF I MOD 5=0 THEN K=4 ELSE K=2
400 LINE (J,0)-(J,K),PSET,P*1.4:LINE (J,180-K)-(J,180),PSET,P*1.
4:NEXT
410 ON P GOTO 420,430,440
420 SCREEN ,,1:LAYER 2,1,4,3:RETURN
430 SCREEN ,,2:LAYER 2,4,1,3:RETURN
440 SCREEN ,,3:LAYER 2,3,4,1:RETURN
450 LABEL "00":LABEL "01":LABEL "02":LABEL "03":LABEL "04":LABEL
 "05":LABEL "06":LABEL "07":LABEL "09":LABEL "0A":LABEL "0D":LAB
EL "0E":LABEL "0F":LABEL "12":LABEL "13":LABEL "15":LABEL "16":L
ABEL "19":LABEL "1A":LABEL "1B"
460 RETURN
470 LABEL "08":GOSUB "1D":IF LEN(A$(P-1,Y))>X*4 THEN  A$(P-1,Y)=
LEFT$(A$(P-1,Y),X*4)+RIGHT$(A$(P-1,Y),LEN(A$(P-1,Y))-(X+1)*4):YY
=Y:XX=X:GOSUB "DISP L":Y=YY:X=XX
480 RETURN
490 LABEL "0B":X=0:Y=0:RETURN
500 LABEL "0C":LOCATE 3,24:INPUT"Are You Sure (Y/N) "+CHR$(5),I$
:IF I$<>"Y" THEN RETURN
510 LABEL "0C1":FOR I=0 TO 9:A$(P-1,I)=ZZ$:NEXT:CLS P:PP=0:GOSUB
 "PAGE":CONSOLE 24,1:X=0:Y=0:RETURN
520 LABEL "10":HCOPY1:HCOPY2:HCOPY3:RETURN
530 LABEL "11":CONSOLE 24,1:LOCATE 4,24:INPUT "CODE ",Z$:A=VAL(Z
$):LOCATE 0,24:PRINT P;CHR$(5);:GOTO "DISP 1"
540 LABEL "14":CONSOLE 24,1:LOCATE 4,24:INPUT "FILE NAME"+CHR$(5
);NA$:OPEN "O",#1,NA$
550 FOR I=0 TO 2:FOR J=0 TO 9:PRINT #1,A$(I,J):NEXT:NEXT:CLOSE:L
OCATE 0,24:PRINT P;:RETURN
560 LABEL "17":CONSOLE 24,1:LOCATE 4,24:INPUT "FILE NAME"+CHR$(5
);NA$:OPEN "I",#1,NA$
570 FOR II=0 TO 2:P=II+1:GOSUB"0C1":FOR JJ=0 TO 9:INPUT #1,A$(II
,JJ):IF A$(II,JJ)<>ZZ$ THEN Y=JJ:GOSUB "DISP L"
580 NEXT:NEXT:CLOSE:P=1:LOCATE 0,24:PRINT P;:X=0:Y=0:GOSUB "PAGE
":RETURN
590 LABEL "18":IF CH=0 THEN CH=1 ELSE CH=0
600 RETURN
610 LABEL "1C":X=X+1:IF X>36 THEN X=0:Y=Y+1:IF Y>9 THEN Y=0:P=P+
1:GOSUB "PAGE"
620 RETURN
630 LABEL "1D":X=X-1:IF X<0 THEN X=36:Y=Y-1:IF Y<0 THEN Y=9:P=P-
1:GOSUB "PAGE"
640 RETURN
650 LABEL "1E":Y=Y-1:IF Y<0 THEN Y=9:P=P-1:GOSUB "PAGE"
660 RETURN
670 LABEL "1F":Y=Y+1:IF Y>9 THEN Y=0:P=P+1:GOSUB "PAGE"
680 RETURN
690 LABEL "DISP L":P1=P:X=0:FOR KK=0 TO 36:A=VAL(MID$(A$(P-1,Y),
KK*4+1,4)):GOSUB "DISP":NEXT:P=P1:RETURN
700 LABEL "START":WIDTH80:DEFINT A-Z:CANVAS 4,4,4:CLS 4
710 DIM KC(1,255),A$(2,9):RESTORE 760
720 FOR I=0 TO 255:READ KC(0,I):NEXT
115                    100010101
```

```
730 FOR I=128 TO 255:READ KC(1,I):KC(1,I-128)=KC(0,I-128):NEXT
740 ZZ$=STRING$(37,"0895"):FOR I=0 TO 2:FOR J=0 TO 9:A$(I,J)=ZZ$
:NEXT:NEXT
750 CLICK OFF:RETURN
760 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
770 DATA 0,0,0,0,0,0,0,0,0,0,0,0,210,211,212,213
780 DATA 000,110,111,184,180,183,185,113,142,143,186,160,104,161
,105,131
790 DATA 316,317,318,319,320,321,322,323,324,325,107,108,167,165
,168,109
800 DATA 187,333,334,335,336,337,338,339,340,341,342,343,344,345
,346,347
810 DATA 348,349,350,351,352,353,354,355,356,357,358,146,179,147
,116,118
820 DATA 114,365,366,367,368,369,370,371,372,373,374,375,376,377
,378,379
830 DATA 380,381,382,383,384,385,386,387,388,389,390,148,135,149
,117,648
840 DATA 1601,1642,1706,1733,1787,1828,2075,2269,2321,2435,2619,
2737,3158,3204,3324,3430
850 DATA 3545,3637,3666,3738,3864,3883,3908,3909,3921,3935,4059,
4152,4226,4261,4364,4403
860 DATA 000,103,154,155,102,106,482,401,403,405,407,409,467,469
,471,435
870 DATA 161,402,404,406,408,410,411,413,415,417,419,421,423,425
,427,429
880 DATA 431,433,436,438,440,442,443,444,445,446,447,450,453,456
,459,462
890 DATA 463,464,465,466,468,470,472,473,474,475,476,477,479,483
,111,112
900 DATA 4419,4429,4446,4473,4491,4529,4569,4588,4660,4665,4704,
4733,0,0,0,0
910 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
920 DATA 115,119,122,123,124,125,126,127,128,129,130,133,134,135
,136,137
930 DATA 138,139,142,143,144,145,150,151,152,153,156,157,158,159
,162,164
940 DATA 000,103,154,155,102,106,582,501,503,505,507,509,567,569
,571,535
950 DATA 161,502,504,506,508,510,511,513,515,517,519,521,523,525
,527,529
960 DATA 531,533,536,538,540,542,543,544,545,546,547,550,553,556
,559,562
970 DATA 563,564,565,566,568,570,572,573,574,575,576,577,579,583
,111,112
980 DATA 166,169,170,171,172,173,174,175,176,177,178,181,182,183
,163,188
990 DATA 189,190,201,202,203,204,205,206,207,208,209,210,211,212
,213,214
```

第２文字は ' で，ASCIIコードを20進数に直すと$00100111_{(2)}$となり，図10－５のBの部分を表わします。

PATTERNは，文字列で与えられたビットパターン(ここではKANJI$(&H7F1))を，画面上に表示する命令です。そして画面上のどこから表示するかを決めるのがPOSITIONです。またPATTERNの第１パラメータで表示していく方向（負のときは下，正のときは上方向）と段数を表わします。

もう１つのポイントは，マルチスクリーンを使っていることです。マルチスクリーン（画面）モードでは，３枚のグラフィック画面（これまでは，R，G，Bの３画面として扱ってきましたが）にそれぞれ８色のうち１色ずつ指定することができます。

〔図10－５〕

```
漢
A 0100000010001000
B 0010011111111111
  0001000010001000
  0000000000000000
  0100001111111110
  0010001000100010
  0001001000100010
  0000001111111110
  0000000000100000
  0000101111111110
  0000100000100000
  0001011111111111
  0001000000100000
  0010000001010000
  0010000110001100
  0100011000000011
```

〔図10－６〕

これを行なっているのが，プログラム700行のCANVAS文です。CANVAS文は，マルチスクリーンモード時に各ページを表示する色を決めるもので，３つのパラメータがそれぞれ１ページから３ページのパレットコードを指定します。CANVAS 4，4，4で３つのグラフィック画面の色をカラーコード４(緑)に指定しています。マルチスクリーンモードのときは，SCREEN文の３つ目のパラメータで，どのページに書き込むのかを指定することになります。

0……指定のパレットコードで書き込む
1……グラフィック１（１ページ）に書き込む
2……グラフィック２（２ページ）に書き込む
3……グラフィック３（３ページ）に書き込む

他のプログラムで、３番目のパラメータを省略しているのは，０を指定したことになり，この場合はLINE文などの色指定が有効になります。１，２，３を指定した場合は，CANVAS文によって，各ページごとに指定した色で書き込まれます。40文字のモードのときは，SCREEN文の１，２番目のパラメータとの組み合わせで，６枚の画面を使うことができます。

画面をマルチスクリーンとして３枚独立に使っていますが，SCREENの３番目のパラメータでモード指定を行なっても，３枚の画面が重なって見えてしまいます。そこで，１枚目と２枚目のグラフィック画面の間に，"■"で塗りつぶしたTEXT画面を挿入します。こうすることで，２枚目と３枚目のグラフィック画面がTEXT画面の影になって，１枚目のグラフィック画面だけが見えるようになります。この表示画面の重ねる順番（優先順位）を決めるのがLAYER文です。

LAYER文は，

LAYER t, g_1, g_2, g_3

のように書き，4つのパラメータがそれぞれテキスト，グラフィック1，グラフィック2，グラフィック3の優先順位に対応します。指定する数値は，1～4までで，同じ数値を2つ使うとエラーになります。この処理はプログラム410行から440行で行なっています。変数Pは表示したいページ番号をさします。

細かい処理としては，450行に未使用のコントロールコードの飛び先を集めて，何もせずにメインルーチンに返していますが，このラベルを使用することで，あらたに機能を追加することができます。

また，作成された文書は，シーケンシャルファイルで記録されますが，データ書き込み，読み出しは，それぞれプログラム540行，560行で行なっています。

その他にもさまざまなテクニックを使っていますが，それはみなさんで研究してください。最後に，変数表を載せますので参考にしてください。

A：JISコード

S：1つ前に入力されたJISコード

A$(P,Y)：PページのY行目のJISコード

X：列

Y：行

P：ページ

B$(0)：入力された文字（キーボード上の）

CH：カタカナ，ひらがなの区別

```
100 '     ｶﾝｲ ﾜｰﾄﾞ ﾌﾟﾛｾｯｻｰ
110 '
120 '                    Programed by T.Watanabe
130 '
140 GOSUB "START"
150 ' MAIN
160 P=1:GOSUB "PAGE"
170 REPEAT:POKE &HEA6,255,255:GOSUB "CURSOR":GOSUB "KEY IN":UNTI
L F
180 MUSIC "O6D0":IF B$(0)<CHR$(32) THEN 210 ELSE S=A:A=KC(CH,ASC
(B$(0)))
190 IF (A=111 OR A= 112) AND S>410 AND S<560 THEN GOSUB "CHANGE"
200 GOSUB "DISP 1":GOTO 170
210 ON ASC(B$(O))+1 GOSUB "00","01","02","03","04","05","06","07
","08","09","0A","0B","0C","0D","0E","0F","10","11","12","13","1
4","15","16","17","18","19","1A","1B","1C","1D","1E","1F":GOTO 1
70
220 LABEL "CHANGE":IF P=1 AND X=0 AND Y=0 THEN RETURN
230 IF X=0 AND Y=0 THEN P=P-1:Y=10
240 IF X=0 THEN Y=Y-1:X=37
250 X=X-1:IF A=111 THEN A=S+1 ELSE A=S+2
260 RETURN
270 LABEL "DISP 1":IF LEN(A$(P-1,Y))/4>X THEN MID$(A$(P-1,Y),X*4
+1,4)=RIGHT$("0000"+RIGHT$(STR$(A),LEN(STR$(A))-1),4) ELSE A$(P-
1,Y)=A$(P-1,Y)+RIGHT$("0000"+RIGHT$(STR$(A),LEN(STR$(A))-1),4)
280 LABEL "DISP":CONSOLE 24,1
290 IF PP<>P THEN GOSUB "PAGE"
```

〈HOUSE〉

```
10 WIDTH40:SCREEN0,0:CLS4
20 WINDOW:GOSUB "HOUSE":GOSUB "FLAME"
30 LOCATE0,23:COLOR7:PRINT"ｶｸﾀﾞｲ --- 1.  ｼｭｸｼｮｳ --- 2 ";
40 REPEAT:Z$=INKEY$(1):UNTIL Z$="1" OR Z$="2"
50 IF Z$="1" GOSUB"*2" ELSE GOSUB"/2"
60 SCREEN0,0:GOTO30
70 LABEL"*2"
80 LOCATE0,23:PRINTCHR$(5):LOCATE0,23:PRINT"SELEST 1 - 4 ";
90 REPEAT:A$=INKEY$(1):UNTIL Z$>"0" AND Z$<"5"
100 SCREEN1,1:CLS4:ON VAL(A$) GOSUB 140,150,160,170:GOSUB "HOUSE
"
110 LOCATE0,20:PRINT"Hit Any KEY"
120 REPEAT:X$=INKEY$:UNTIL X$<>""
130 RETURN
140 WINDOW(0,0)-(319,199),(0,0)-(160,100):RETURN
150 WINDOW(0,0)-(319,199),(160,0)-(319,100):RETURN
160 WINDOW(0,0)-(319,199),(0,100)-(160,199):RETURN
170 WINDOW(0,0)-(319,199),(160,100)-(319,199):RETURN
180 LABEL"/2"
190 LOCATE0,23:PRINTCHR$(5):LOCATE0,23:PRINT"nﾌﾞﾝ ﾉ ｲﾁ ? 2 - 6 "
;
200 REPEAT:A$=INKEY$(1):UNTIL VAL(A$)>1 AND VAL(A$)<7
210 N=VAL(A$)
220 SCREEN1,1:CLS4:WINDOW(0,0)-(319/N,199/N),(0,0)-(319,199):GOS
UB "HOUSE"
230 LOCATE0,20:PRINT"Hit Any KEY"
240 REPEAT:X$=INKEY$:UNTIL X$<>""
250 RETURN
260 END
270 LABEL"HOUSE"
280 LINE(100, 90)-(220,170),PSET,4,BF
290 COLOR2:LINE(120,40)-(200,40)-(240, 90)-(80, 90)-(120,40)
300 PAINT(125,45),HEXCHR$("00FF1100FF44"),2
310 PAINT(195,45),HEXCHR$("00FF1100FF44"),2
320 LINE(120,110)-(170,140),PSET,7,B
330 LINE(121,111)-(169,139),PSET,0,BF
340 LINE(120,125)-(170,125),PSET,7
350 LINE(145,110)-(145,140),PSET,7
360 LINE(180,110)-(210,170),PSET,1,BF
370 LINE(130,20)-(150,60),PSET,6,BF:COLOR7
380 RETURN
390 LABEL"FLAME"
400 LINE(0,0)-(319,176),PSET,5,B
410 LINE(160,0)-(160,176),PSET,5:LINE(0,100)-(319,100),PSET,5
420 LOCATE0,1:PRINT"1":LOCATE20,1:PRINT"2":LOCATE0,13:PRINT"3":L
OCATE20,13:PRINT"4"
430 RETURN
```

〈テ　マ　リ〉

```
10 WIDTH80:SCREEN0,0:CLS4
20 C=3
50 XA=.5:YA=.6:ZA=0:R=120
190 FOR XYZ=1 TO 3
200 FOR SS=0 TO 180 STEP 45
210 FOR S=SS-8 TO SS+8 STEP 4
220 G=0:READ C
230 FOR TH=0 TO PAI(2) STEP PAI(1)/24
240 ON XYZ GOSUB"X","Y","Z"
250 GOSUB"ROTA"
260 IF Z3<0 THEN G=0:GOTO 280
270 GOSUB"LINE"
280 NEXT TH
290 NEXT S:NEXT SS:NEXT XYZ
295 CIRCLE(320,100),60,7
300 END
310 DATA 6,6,4,6,6, 7,5,3,5,7, 5,3,2,3,5, 4,4,5,4,4, 2,2,4,2,2
320 DATA 5,1,5,1,5, 6,4,6,4,6, 2,2,5,2,2, 4,4,6,4,4, 7,6,4,6,7
330 DATA 7,1,2,1,7, 4,4,5,4,4, 6,4,6,4,6, 5,3,2,3,5, 5,1,5,1,5
500 LABEL"X"
510 P=RAD(S)
520 X=R*SIN(TH)*COS(P):Y=R*COS(TH):Z=R*SIN(TH)*SIN(P)
530 RETURN
540 LABEL"Y"
550 P=RAD(S)
560 Y=R*SIN(TH)*COS(P):X=R*COS(TH):Z=R*SIN(TH)*SIN(P)
570 RETURN
580 LABEL"Z"
590 P=RAD(S)
600 Y=R*SIN(TH)*COS(P):Z=R*COS(TH):X=R*SIN(TH)*SIN(P)
610 RETURN
620 LABEL"ROTA"
630 X1=X
640 Y1=Y*COS(XA)-Z*SIN(XA)
650 Z1=Y*SIN(XA)+Z*COS(XA)
660 X2=X1*COS(YA)-Z1*SIN(YA)
670 Y2=Y1
680 Z2=X1*SIN(YA)+Z1*COS(YA)
690 X3=X2*COS(ZA)-Y2*SIN(ZA)
700 Y3=X2*SIN(ZA)+Y2*COS(ZA)
710 Z3=Z2
720 RETURN
730 LABEL"LINE"
740 GX=X3+320:GY=100-.5*Y3
750 IF GX<0 OR GX>639 THEN G=0:RETURN
760 IF GY<0 OR GY>199 THEN G=0:RETURN
770 IF G=1 GOTO790
780 LINE(GX,GY)-(GX,GY),PSET,C
790 LINE-(GX,GY),PSET,C
800 G=1
810 RETURN
```

〈百人一首〉

```
10 WIDTH40:SCREEN1,1:CLS4:LOCATE2,12:CSIZE3:CFLASH1:COLOR6:PRINT
#0"ｼｮｳｼｮｳ ｵﾏﾁｸﾀﾞｻｲ｡"
100 Q$="FFFFFFFFFFFFFFFF000000000000000055AA55AA55AA55AA"
101 FOR I=1 TO 54:READ A:DEFCHR$(A)=HEXCHR$(Q$):NEXT
102 DATA 0,1,6,7,8,9,10,16,17,23,24,25,26,33,34,40,41,42,48,49,5
7,58,59,60,61,62,63,64,65
103 DATA 74,75,76,77,78,79,80,92,93,94,95,109,110,111,143,144,14
5,146,147,148,149,150,157,158,159
104 Q$="FFFFFFFFFFFFFEFE0000000000000000"
105 DEFCHR$(2)=HEXCHR$(Q$+"55AA55AA55AA54AA")
106 Q$="FFF8E0C08040A0540000000000060F0FC"
107 DEFCHR$(3)=HEXCHR$(Q$+"55A840800060F0FC")
108 Q$="FF0F0301000000000000000000000000"
109 DEFCHR$(4)=HEXCHR$(Q$+"550A010000000000")
110 Q$="FFFFFFFFFF7F3F3F0000000000000000"
111 DEFCHR$(5)=HEXCHR$(Q$+"55AA55AA552A152A")
122 Q$="EAE5EAE5EAD5CAD5EFEFEFEFEFDFDFDF"
123 DEFCHR$(11)=HEXCHR$(Q$+"EFEFEFEFEFDFDFDF")
124 Q$="CAD5AA95AA952A55DFDFBFBFBFBF7F7F"
125 DEFCHR$(12)=HEXCHR$(Q$+"DFDFBFBFBFBF7F7F")
126 Q$="2A552A552A552A557F7F7F7F7F7F7F7F"
127 DEFCHR$(13)=HEXCHR$(Q$+"7F7F7F7F7F7F7F7F")
128 Q$="2A442A442A442A846E6E6E6E6E6E6EAE"
129 DEFCHR$(14)=HEXCHR$(Q$+"6E6E6E6E6E6E6EAE")
130 Q$="EAE4EAE4EAF4FAFFEEEEEEEEEEEF4FAFF"
131 DEFCHR$(15)=HEXCHR$(Q$+"EEEEEEEEEEF4FAFF")
136 Q$="FEFEFFFFFFFFFFFF0000000000000000"
137 DEFCHR$(18)=HEXCHR$(Q$+"54AA55AA55AA55AA")
138 Q$="45AA518255EAD5EACFFFF343373F3F3F"
139 DEFCHR$(19)=HEXCHR$(Q$+"CFFFF3C377BF7FBD")
140 Q$="000080008040A85700008080C0E0F8FF"
141 DEFCHR$(20)=HEXCHR$(Q$+"00008080C0E0F8FF")
142 Q$="1F0F0703010026C700000000000028D0"
143 DEFCHR$(21)=HEXCHR$(Q$+"150A0502010038F8")
144 Q$="FFFFFFFFFFFF7F3F0000000000000000"
145 DEFCHR$(22)=HEXCHR$(Q$+"55AA55AA55AA552A")
154 Q$="AA55AA55AA55AA55FFFFFFFFFFFFFFFF"
155 DEFCHR$(27)=HEXCHR$(Q$+"FFFFFFFFFFFFFFFF")
156 Q$="AA55AA55AA55AA55FFFFFFFFFFFFFFFF"
157 DEFCHR$(28)=HEXCHR$(Q$+"FFFFFFFFFFFFFFFF")
158 Q$="AA55AA55AA55AA55FFFFFFFFFFFFFFFF"
159 DEFCHR$(29)=HEXCHR$(Q$+"FFFFFFFFFFFFFFFF")
160 Q$="AA55AA45A245A245FFFFEFEFE7E7E7E7"
161 DEFCHR$(30)=HEXCHR$(Q$+"FFFFEFEFE7E7E7E7")
162 Q$="A245A245A248AD5FE7E7E7E7E7EAED5F"
163 DEFCHR$(31)=HEXCHR$(Q$+"E7E7E7E7E7EAED5F")
164 Q$="00000000000000000000000000000000"
165 DEFCHR$(32)=HEXCHR$(Q$+"0000000000000000")
170 Q$="FBFDFFFFFFFF0E0C0070301000000D1B3B"
171 DEFCHR$(35)=HEXCHR$(Q$+"51AB57AA55A04080")
172 Q$="55AA7AE080000000FFFF8E1F7FFFFFFF"
173 DEFCHR$(36)=HEXCHR$(Q$+"FFFFDFA300000000")
174 Q$="8F1F3C78F0600000A04083070F9FDCE0"
175 DEFCHR$(37)=HEXCHR$(Q$+"F0E0C0800000030F")
176 Q$="1F0F070301000000C0E0E0C00000C3C"
177 DEFCHR$(38)=HEXCHR$(Q$+"150A051239F8F0C0")
178 Q$="FFFFFFFFFFFF7F3F0000000000000000"
179 DEFCHR$(39)=HEXCHR$(Q$+"55AA55AA55AA552A")
186 Q$="7F7F7F3FBF3FBF3F7F7F7FBFBFBFBFBF"
187 DEFCHR$(43)=HEXCHR$(Q$+"7F7F7FBFBFBFBFBF")
```

```
188 Q$="BF5F9F4FAF57A757BFDFDFEFEFF7F7F7"
189 DEFCHR$(44)=HEXCHR$(Q$+"BFDFDFEFEFF7F7F7")
190 Q$="AB53A91529544A64FBFBFDBD3D5E5E6E"
191 DEFCHR$(45)=HEXCHR$(Q$+"FBFBFDBD3D5E5E6E")
192 Q$="6A74797F7F7F7F7F6E76797F7F7F7F7F"
193 DEFCHR$(46)=HEXCHR$(Q$+"6E76797F7F7F7F7F")
194 Q$="7F7F7F7F7F7FFFFF7F7F7F7F7F7FFFFF"
195 DEFCHR$(47)=HEXCHR$(Q$+"7F7F7F7F7F7FFFFF")
200 Q$="FFFFFFFEFCF8F1F20000000103070F0F"
201 DEFCHR$(50)=HEXCHR$(Q$+"55AA55AA54A851A2")
202 Q$="C3830101010000003B7BFDFDFEFEFE"
203 DEFCHR$(51)=HEXCHR$(Q$+"4383010101000000")
204 Q$="303B1F8DC0606060FFFFFFFFFFFFFFFF"
205 DEFCHR$(52)=HEXCHR$(Q$+"302B168DC0E0E0E0")
206 Q$="000080C0C0000000F0F9FDFEFFFFFFFF"
207 DEFCHR$(53)=HEXCHR$(Q$+"07028040C0000000")
208 Q$="0000000000000000FEFEFFFF7FBFDFEF"
209 DEFCHR$(54)=HEXCHR$(Q$+"0000000000000000")
210 Q$="1F0F0703010000000000008080C0E0E0"
211 DEFCHR$(55)=HEXCHR$(Q$+"150A050201000000")
212 Q$="FFFFFFFFFFFF7F3F0000000000000000"
213 DEFCHR$(56)=HEXCHR$(Q$+"55AA55AA55AA552A")
232 Q$="E0C0C0E0E0E0E01F3F3F1F1F1F1F1F"
233 DEFCHR$(66)=HEXCHR$(Q$+"448840A040A142A0")
234 Q$="0000000000000000FFFFFFFFFFFFFFFF"
235 DEFCHR$(67)=HEXCHR$(Q$+"1010202040800000")
236 Q$="7030303818181C1C7FBFBFBF9FDFDFDF"
237 DEFCHR$(68)=HEXCHR$(Q$+"70B0B0B898D8DCDC")
238 Q$="0000000000100804FFFFFFFFFFFFFFFF"
239 DEFCHR$(69)=HEXCHR$(Q$+"0000000000201008")
240 Q$="0000000000000000F7F7FBFBFBFDFDFE"
241 DEFCHR$(70)=HEXCHR$(Q$+"0000000000000000")
242 Q$="0000000000000000F0F0F8F8FCFCFEFE"
243 DEFCHR$(71)=HEXCHR$(Q$+"0000000000000000")
244 Q$="1F0F030100000000000000000000000"
245 DEFCHR$(72)=HEXCHR$(Q$+"150A010000000000")
246 Q$="FFFFFFFF7F3F0F070000000000000000"
247 DEFCHR$(73)=HEXCHR$(Q$+"55AA55AA552A0502")
262 Q$="FFFFFFFFFEF8F0E000000000001070F1F"
263 DEFCHR$(81)=HEXCHR$(Q$+"55AA55AA54A850A0")
264 Q$="C0C08000000000003F3F7FFFFFFFFFFF"
265 DEFCHR$(82)=HEXCHR$(Q$+"40861820C0000000")
266 Q$="0000000000000000FFFFFFFFFFFFFEFE"
267 DEFCHR$(83)=HEXCHR$(Q$+"000000010307060E")
268 Q$="1C1C1C1C1C1C1C1CDFDFDFDF9F5FDFDF"
269 DEFCHR$(84)=HEXCHR$(Q$+"DCDCDCDC9E5EDFDF")
270 Q$="0000000000000000FFFFFFFFFFFFFFFF"
271 DEFCHR$(85)=HEXCHR$(Q$+"0000000000000080")
272 Q$="0000000000000000FEFFFFFFFFFFFFFF"
273 DEFCHR$(86)=HEXCHR$(Q$+"0000000000000000")
274 Q$="00000000000000007F7FBFBFDFEFF7F9"
275 DEFCHR$(87)=HEXCHR$(Q$+"0000000000000000")
276 Q$="00000000000000008080C0C0E0F0F0"
277 DEFCHR$(88)=HEXCHR$(Q$+"0000000000000000")
278 Q$="03010000000000000000000000000000"
279 DEFCHR$(89)=HEXCHR$(Q$+"0100000000000000")
280 Q$="FFFF7F1F0F0701000000000000000000"
281 DEFCHR$(90)=HEXCHR$(Q$+"55AA550A05020100")
282 Q$="FFFFFFFFFFFFFF7F0000000000000000"
283 DEFCHR$(91)=HEXCHR$(Q$+"55AA55AA55AA552A")
292 Q$="FFFFFFFFFFFFFEFD0000000000010303"
293 DEFCHR$(96)=HEXCHR$(Q$+"55AA55AA55AB57AB")
294 Q$="E0C0808050AA5ABE1F3F7F7FFFF3E3E3"
```

```
295 DEFCHR$(97)=HEXCHR$(Q$+"4080008070FEF6EA")
296 Q$="0000000000000000FFFFFFFFFFFFFFFF"
297 DEFCHR$(98)=HEXCHR$(Q$+"0000000000000000")
298 Q$="0103030307070707FDFBFBFBF7F7F7F7"
299 DEFCHR$(99)=HEXCHR$(Q$+"1D3B3B3B77767676")
300 Q$="D8DCDCDCDCEEEEEEFDFDFDFDFDFEFEFEF"
301 DEFCHR$(100)=HEXCHR$(Q$+"1F1F1F1F1F0F0F0F")
302 Q$="000000205028140AFFFFFFFFFFFFFFFF"
303 DEFCHR$(101)=HEXCHR$(Q$+"80C0709F8FC7C3E0")
304 Q$="0000000000000000FFFFFFFFFFFFFFFF"
305 DEFCHR$(102)=HEXCHR$(Q$+"000000F0FFFFFFFF")
306 Q$="0000000000000000FEFFFFFFFFFFFFFF"
307 DEFCHR$(103)=HEXCHR$(Q$+"00000000FCFFFFFF")
308 Q$="000000000000000078B8CCF2FFFFFFFF"
309 DEFCHR$(104)=HEXCHR$(Q$+"000000000080E0F0")
310 Q$="000000000000000000000000080C0E0F8"
311 DEFCHR$(105)=HEXCHR$(Q$+"0000000000000000")
312 Q$="00000000000000000000000100C0703"
313 DEFCHR$(106)=HEXCHR$(Q$+"0000000000000000")
314 Q$="1F0700000000000000000000000038FFF"
315 DEFCHR$(107)=HEXCHR$(Q$+"1502000000000000")
316 Q$="FFFFFF7F3F0F030000000000C0F0FCFF"
317 DEFCHR$(108)=HEXCHR$(Q$+"55AA552A150A0100")
324 Q$="FAFDFAFFFAFFFFFF0707070505050000"
325 DEFCHR$(112)=HEXCHR$(Q$+"57AF57AF55AF55AA")
326 Q$="B87CBCFCBEFEFEFEE7C7474743038303"
327 DEFCHR$(113)=HEXCHR$(Q$+"F4EC54EC56AAD6AA")
328 Q$="00000000000000007F9CE3FFFFFFFFFF"
329 DEFCHR$(114)=HEXCHR$(Q$+"0000000000000000")
330 Q$="070F0F0F0F0F0707F7EFEFEFEFEFEFF7F7"
331 DEFCHR$(115)=HEXCHR$(Q$+"366C6C6C6C6C7636")
332 Q$="EFEFEFF7F7F7F7F7EFEFEFF7F7F7F7F7"
333 DEFCHR$(116)=HEXCHR$(Q$+"0F0F0F0707070707")
334 Q$="150A050285C2C1E0FFFFFFFFFFFFFFFF"
335 DEFCHR$(117)=HEXCHR$(Q$+"E0F0F0F8F8FCFCFE")
336 Q$="40AA55AA55AA55AAFFFFFFFFFFFFFFFF"
337 DEFCHR$(118)=HEXCHR$(Q$+"3F00000000000000")
338 Q$="00A055AA55AA55AAFFFFFFFFFFFFFFFF"
339 DEFCHR$(119)=HEXCHR$(Q$+"FF0F000000000000")
340 Q$="000040A854AA55AAFFFFFFFFFFFFFFFF"
341 DEFCHR$(120)=HEXCHR$(Q$+"F8FC3E0701000000")
342 Q$="000000000000000080FCFEFFFFFFFFFFFF"
343 DEFCHR$(121)=HEXCHR$(Q$+"0000000080804020")
344 Q$="000000000000000000030FF7FBFCFFFF"
345 DEFCHR$(122)=HEXCHR$(Q$+"0000000000000000")
346 Q$="000000003070F7F7FBFEFFBFEFD7AD5"
347 DEFCHR$(123)=HEXCHR$(Q$+"0000000002050A55")
348 Q$="0000000080C0F0FCFFFDEAD5AA55AA55"
349 DEFCHR$(124)=HEXCHR$(Q$+"00031F3FFF7FAF57")
350 Q$="FF3F0700000000000040A855AA55AA55"
351 DEFCHR$(125)=HEXCHR$(Q$+"55EAFDFFFFFFFFFF")
352 Q$="FFFFFFFF1F010000000000A054AA55"
353 DEFCHR$(126)=HEXCHR$(Q$+"55AA55AAF5FEFFFF")
354 Q$="FFFFFFFFFFFF3F0F000000000008050"
355 DEFCHR$(127)=HEXCHR$(Q$+"55AA55AA55AAD5FA")
356 Q$="FFFFFFFFFFFFFFFF0F0F1F1F1F1F0000"
357 DEFCHR$(128)=HEXCHR$(Q$+"50A74FAF4FA055AA")
358 Q$="FEFFFFFFFFFFFFFFE3F1F0F8F8F80000"
359 DEFCHR$(129)=HEXCHR$(Q$+"16EBE5F2F50255AA")
360 Q$="00008080C0F1FFFFFFFFFFFF7F0E0000"
361 DEFCHR$(130)=HEXCHR$(Q$+"0000808040A055AA")
362 Q$="001038387CFEFFFFFEFEFC7C783010000"
363 DEFCHR$(131)=HEXCHR$(Q$+"0F0F172F57AB55AA")
```

```
364 Q$="E34307070F0F1FFFEF5DB9F8F0F0E000"
365 DEFCHR$(132)=HEXCHR$(Q$+"0F1FBDFAF5FAF5AA")
366 Q$="C0E0F0F8FCFEFFFFFFFFFFFF7F3F1F00"
367 DEFCHR$(133)=HEXCHR$(Q$+"FFFFFFFF7FBF5FAA")
368 Q$="552A15020000FCFEFFFFFFFFFFFF0301"
369 DEFCHR$(134)=HEXCHR$(Q$+"80C0E0F8FFFF54AA")
370 Q$="55AA55AA15000000FFFFFFFFFFFFFFFF"
371 DEFCHR$(135)=HEXCHR$(Q$+"00000000C0FFFF00")
372 Q$="55AA55AA550A0000FFFFFFFFFFFFFFFF"
373 DEFCHR$(136)=HEXCHR$(Q$+"0000000000E0FF00")
374 Q$="40A040A040000107FFFFFFFFFFFFFEF8"
375 DEFCHR$(137)=HEXCHR$(Q$+"1010101020408102")
376 Q$="00000000000080C0FFFFFF7EB552A15"
377 DEFCHR$(138)=HEXCHR$(Q$+"0102050E3FFF7FBF")
378 Q$="0701000000000000FAFFFFFFFF5FAA55"
379 DEFCHR$(139)=HEXCHR$(Q$+"52AB55AAD5FAFFFF")
380 Q$="FCFEFFFF3F0F0300AA55AA55EAF5FEFF"
381 DEFCHR$(140)=HEXCHR$(Q$+"AB55AA556AA556AA")
382 Q$="0000E0FFFFFFFF7FAA55AA55AA50A080"
383 DEFCHR$(141)=HEXCHR$(Q$+"FFFFBF55AB52B52A")
384 Q$="00071F7FFFFFFFFFAA50A00000000000"
385 DEFCHR$(142)=HEXCHR$(Q$+"FFFAF5AA55AA55AA")
402 Q$="80C0FFFFFFFFFFFF7F3F000000000000"
403 DEFCHR$(151)=HEXCHR$(Q$+"008055AA55AA55AA")
404 Q$="0000FFFFFFFFFFFFFFFF000000000000"
405 DEFCHR$(152)=HEXCHR$(Q$+"000055AA55AA55AA")
406 Q$="3F7FFFFFFFFFFFFFC080000000000000"
407 DEFCHR$(153)=HEXCHR$(Q$+"152A55AA55AA55AA")
408 Q$="E0F0F8FCFFFFFFFF0A05020100000000"
409 DEFCHR$(154)=HEXCHR$(Q$+"5FAF57AB55AA55AA")
410 Q$="00000003FFFFFFFFAA55AA5400000000"
411 DEFCHR$(155)=HEXCHR$(Q$+"FFFFFFFE55AA55AA")
412 Q$="00010F7FFFFFFFFF7FAE508000000000"
413 DEFCHR$(156)=HEXCHR$(Q$+"D5F4F5AA55AA55AA")
418 Q$="FFFFFFFFFFFFFFFF0000000000000000"
419 DEFCHR$(159)=HEXCHR$(Q$+"55AA55AA55AA55AA")
422 Q$="00000000609060000000000060906000"
423 DEFCHR$(161)=HEXCHR$(Q$+"0000000060906000")
424 Q$="007C404040400000007C404040400000"
425 DEFCHR$(162)=HEXCHR$(Q$+"007C404040400000")
426 Q$="000020202023E000000020202023E00"
427 DEFCHR$(163)=HEXCHR$(Q$+"0000202020023E00")
428 Q$="00000000004020300000000000402030"
429 DEFCHR$(164)=HEXCHR$(Q$+"0000000000402030")
430 Q$="FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"
431 DEFCHR$(165)=HEXCHR$(Q$+"FFFFFFFFFFFFFFFF")
432 Q$="20F0243814203C0020F0243814203C00"
433 DEFCHR$(166)=HEXCHR$(Q$+"20F0243814203C00")
434 Q$="00107E147E5A360000107E147E5A3600"
435 DEFCHR$(167)=HEXCHR$(Q$+"00107E147E5A3600")
436 Q$="7C087C92A6AA4C007C087C92A6AA4C00"
437 DEFCHR$(168)=HEXCHR$(Q$+"7C087C92A6AA4C00")
438 Q$="001C001C02020C00001C001C02020C00"
439 DEFCHR$(169)=HEXCHR$(Q$+"001C001C02020C00")
440 Q$="38103C42121C2A4538103C42121C2A45"
441 DEFCHR$(170)=HEXCHR$(Q$+"38103C42121C2A45")
442 Q$="00103C121C32360000103C121C323600"
443 DEFCHR$(171)=HEXCHR$(Q$+"00103C121C323600")
444 Q$="0004243C520A08000004243C520A0800"
445 DEFCHR$(172)=HEXCHR$(Q$+"0004243C520A0800")
446 Q$="0008085E6A4E08000008085E6A4E0800"
447 DEFCHR$(173)=HEXCHR$(Q$+"0008085E6A4E0800")
448 Q$="00080E08182C1A0000080E08182C1A00"
```

```
449 DEFCHR$(174)=HEXCHR$(Q$+"00080E08182C1A00")
450 Q$="0000001C62020C000000001C62020C00"
451 DEFCHR$(175)=HEXCHR$(Q$+"0000001C62020C00")
452 Q$="0000007C000000000000007C00000000"
453 DEFCHR$(176)=HEXCHR$(Q$+"0000007C00000000")
454 Q$="20FC287CAA92640020FC287CAA926400"
455 DEFCHR$(177)=HEXCHR$(Q$+"20FC287CAA926400")
456 Q$="00000482825030000000048282503000"
457 DEFCHR$(178)=HEXCHR$(Q$+"0000048282503000")
458 Q$="1C003C42020418001C003C4202041800"
459 DEFCHR$(179)=HEXCHR$(Q$+"1C003C4202041800")
460 Q$="38007C081038460038007C0810384600"
461 DEFCHR$(180)=HEXCHR$(Q$+"38007C0810384600")
462 Q$="20F422207CA2640020F422207CA26400"
463 DEFCHR$(181)=HEXCHR$(Q$+"20F422207CA26400")
464 Q$="2024F22A284810002024F22A28481000"
465 DEFCHR$(182)=HEXCHR$(Q$+"2024F22A28481000")
466 Q$="103C083E04201C00103C083E04201C00"
467 DEFCHR$(183)=HEXCHR$(Q$+"103C083E04201C00")
468 Q$="08081060100808000808106010080800"
469 DEFCHR$(184)=HEXCHR$(Q$+"0808106010080800")
470 Q$="04445E444444080004445E4444440800"
471 DEFCHR$(185)=HEXCHR$(Q$+"04445E4444440800")
472 Q$="0038040800403C000038040800403C00"
473 DEFCHR$(186)=HEXCHR$(Q$+"0038040800403C00")
474 Q$="100A3C0402201C00100A3C0402201C00"
475 DEFCHR$(187)=HEXCHR$(Q$+"100A3C0402201C00")
476 Q$="201020202221C00201020202221C00"
477 DEFCHR$(188)=HEXCHR$(Q$+"201020202221C00")
478 Q$="08FE38483808300008FE384838083000"
479 DEFCHR$(189)=HEXCHR$(Q$+"08FE384838083000")
480 Q$="2424FE2420201E002424FE2420201E00"
481 DEFCHR$(190)=HEXCHR$(Q$+"2424FE2420201E00")
482 Q$="2418307E040804002418307E04080400"
483 DEFCHR$(191)=HEXCHR$(Q$+"2418307E04080400")
484 Q$="20F04E5280908E0020F04E5280908E00"
485 DEFCHR$(192)=HEXCHR$(Q$+"20F04E5280908E00")
486 Q$="087E103C62021C00087E103C62021C00"
487 DEFCHR$(193)=HEXCHR$(Q$+"087E103C62021C00")
488 Q$="00986402020418000098640202041800"
489 DEFCHR$(194)=HEXCHR$(Q$+"0098640202041800")
490 Q$="007E040810100C00007E040810100C00"
491 DEFCHR$(195)=HEXCHR$(Q$+"007E040810100C00")
492 Q$="0044483020427C000044483020427C00"
493 DEFCHR$(196)=HEXCHR$(Q$+"0044483020427C00")
494 Q$="20F44A881C2A180020F44A881C2A1800"
495 DEFCHR$(197)=HEXCHR$(Q$+"20F44A881C2A1800")
496 Q$="004E424040482F00004E424040482F00"
497 DEFCHR$(198)=HEXCHR$(Q$+"004E424040482F00")
498 Q$="48287C92A6AA460048287C92A6AA4600"
499 DEFCHR$(199)=HEXCHR$(Q$+"48287C92A6AA4600")
500 Q$="4048F4644CD64C004048F4644CD64C00"
501 DEFCHR$(200)=HEXCHR$(Q$+"4048F4644CD64C00")
502 Q$="00385492926204000038549292620400"
503 DEFCHR$(201)=HEXCHR$(Q$+"0038549292620400")
504 Q$="445E44444C544A00445E44444C544A00"
505 DEFCHR$(202)=HEXCHR$(Q$+"445E44444C544A00")
506 Q$="2064A624242418002064A62424241800"
507 DEFCHR$(203)=HEXCHR$(Q$+"2064A62424241800")
508 Q$="30003048084A920030003048084A9200"
509 DEFCHR$(204)=HEXCHR$(Q$+"30003048084A9200")
510 Q$="00102844020200000010284402020000"
511 DEFCHR$(205)=HEXCHR$(Q$+"0010284402020000")
```

```
512 Q$="9E849E848C964C009E849E848C964C00"
513 DEFCHR$(206)=HEXCHR$(Q$+"9E849E848C964C00")
514 Q$="087E083C083C4A30087E083C083C4A30"
515 DEFCHR$(207)=HEXCHR$(Q$+"087E083C083C4A30")
516 Q$="7010147CAAC810007010147CAAC81000"
517 DEFCHR$(208)=HEXCHR$(Q$+"7010147CAAC81000")
518 Q$="24F22060A2623C0024F22060A2623C00"
519 DEFCHR$(209)=HEXCHR$(Q$+"24F22060A2623C00")
520 Q$="08487CAA9292640008487CAA92926400"
521 DEFCHR$(210)=HEXCHR$(Q$+"08487CAA92926400")
522 Q$="1078207820221C001078207820221C00"
523 DEFCHR$(211)=HEXCHR$(Q$+"1078207820221C00")
524 Q$="44447CE22620100044447CE226201000"
525 DEFCHR$(212)=HEXCHR$(Q$+"44447CE226201000")
526 Q$="0888BCCACA9C08000888BCCACA9C0800"
527 DEFCHR$(213)=HEXCHR$(Q$+"0888BCCACA9C0800")
528 Q$="18080E08384E380018080E08384E3800"
529 DEFCHR$(214)=HEXCHR$(Q$+"18080E08384E3800")
530 Q$="1804103C62021C001804103C62021C00"
531 DEFCHR$(215)=HEXCHR$(Q$+"1804103C62021C00")
532 Q$="24242424240408002424242424040800"
533 DEFCHR$(216)=HEXCHR$(Q$+"2424242424040800")
534 Q$="3C081C620E121C003C081C620E121C00"
535 DEFCHR$(217)=HEXCHR$(Q$+"3C081C620E121C00")
536 Q$="10147C345414121010147C3454141210"
537 DEFCHR$(218)=HEXCHR$(Q$+"10147C3454141210")
538 Q$="3C08103C42021C003C08103C42021C00"
539 DEFCHR$(219)=HEXCHR$(Q$+"3C08103C42021C00")
540 Q$="2020EC3262A224002020EC3262A22400"
541 DEFCHR$(220)=HEXCHR$(Q$+"2020EC3262A22400")
542 Q$="0808102070CA86000808102070CA8600"
543 DEFCHR$(221)=HEXCHR$(Q$+"0808102070CA8600")
544 Q$="20904000000000002090400000000000"
545 DEFCHR$(222)=HEXCHR$(Q$+"2090400000000000")
546 Q$="60906000000000006090600000000000"
547 DEFCHR$(223)=HEXCHR$(Q$+"6090600000000000")
548 Q$="0000000000000000FFFFFFFF00000000"
549 DEFCHR$(224)=HEXCHR$(Q$+"AA55AA5500000000")
550 Q$="100F0700311F1818100F0700311F1818"
551 DEFCHR$(225)=HEXCHR$(Q$+"100F0700311F1818")
552 Q$="181F181818181F10181F181818181F10"
553 DEFCHR$(226)=HEXCHR$(Q$+"181F181818181F10")
554 Q$="00000203030303030000020303030303"
555 DEFCHR$(227)=HEXCHR$(Q$+"0000020303030303")
556 Q$="0307070D0C1830400307070D0C183040"
557 DEFCHR$(228)=HEXCHR$(Q$+"0307070D0C183040")
558 Q$="000000000000201F000000000000201F"
559 DEFCHR$(229)=HEXCHR$(Q$+"000000000000201F")
560 Q$="0F000000000000000F00000000000000"
561 DEFCHR$(230)=HEXCHR$(Q$+"0F00000000000000")
562 Q$="1008241F07301F181008241F07301F18"
563 DEFCHR$(231)=HEXCHR$(Q$+"1008241F07301F18")
564 Q$="181F181F18181F18181F181F18181F18"
565 DEFCHR$(232)=HEXCHR$(Q$+"181F181F18181F18")
566 Q$="00000000000000000000000000000000"
567 DEFCHR$(233)=HEXCHR$(Q$+"0000000000000000")
568 Q$="FFFFFFFFF7FFFFFF081CDDFF7E3EFFE7"
569 DEFCHR$(234)=HEXCHR$(Q$+"55A2000009800008")
570 Q$="00000000000000000000000000000000"
571 DEFCHR$(235)=HEXCHR$(Q$+"0000000000000000")
572 Q$="00000000000000000000000000000000"
573 DEFCHR$(236)=HEXCHR$(Q$+"0000000000000000")
574 Q$="00000000000000000000000000000000"
```

```
575 DEFCHR$(237)=HEXCHR$(Q$+"0000000000000000")
576 Q$="00000000000000000000000000000000"
577 DEFCHR$(238)=HEXCHR$(Q$+"0000000000000000")
578 Q$="00000000000000000000000000000000"
579 DEFCHR$(239)=HEXCHR$(Q$+"0000000000000000")
580 Q$="0000000000000000FFFFFFFF00000000"
581 DEFCHR$(240)=HEXCHR$(Q$+"AA55AA5500000000")
582 Q$="F0F86CC004FE0C0CF0F86CC004FE0C0C"
583 DEFCHR$(241)=HEXCHR$(Q$+"F0F86CC004FE0C0C")
584 Q$="0CFC0C0C0C0CFC080CFC0C0C0C0CFC08"
585 DEFCHR$(242)=HEXCHR$(Q$+"0CFC0C0C0C0CFC08")
586 Q$="00000000000000000000000000000000"
587 DEFCHR$(243)=HEXCHR$(Q$+"0000000000000000")
588 Q$="00000080C0703C1E00000080C0703C1E"
589 DEFCHR$(244)=HEXCHR$(Q$+"00000080C0703C1E")
590 Q$="00000000000078FC00000000000078FC"
591 DEFCHR$(245)=HEXCHR$(Q$+"00000000000078FC")
592 Q$="06000000000000000600000000000000"
593 DEFCHR$(246)=HEXCHR$(Q$+"0600000000000000")
594 Q$="3020FCFE20C8FC0C3020FCFE20C8FC0C"
595 DEFCHR$(247)=HEXCHR$(Q$+"3020FCFE20C8FC0C")
596 Q$="0CFC0CFC0C0CFC0C0CFC0CFC0C0CFC0C"
597 DEFCHR$(248)=HEXCHR$(Q$+"0CFC0CFC0C0CFC0C")
598 Q$="00000000000000000000000000000000"
599 DEFCHR$(249)=HEXCHR$(Q$+"0000000000000000")
600 Q$="FFFFFBF7FFFFFFFF00001C3C3C380000"
601 DEFCHR$(250)=HEXCHR$(Q$+"55AA458A418255AA")
602 Q$="00000000000000000000000000000000"
603 DEFCHR$(251)=HEXCHR$(Q$+"0000000000000000")
604 Q$="00000000000000000000000000000000"
605 DEFCHR$(252)=HEXCHR$(Q$+"0000000000000000")
606 Q$="00000000000000000000000000000000"
607 DEFCHR$(253)=HEXCHR$(Q$+"0000000000000000")
608 Q$="00000000000000000000000000000000"
609 DEFCHR$(254)=HEXCHR$(Q$+"0000000000000000")
610 Q$="00000000000000000000000000000000"
611 DEFCHR$(255)=HEXCHR$(Q$+"0000000000000000")
902 DEFCHR$(243)=STRING$(24,0)
903 DEFCHR$(160)=STRING$(24,0)
904 DEFCHR$(&HFA)=HEXCHR$("FFFFFBF7FFFFFFFF00001C3C3C38000055AA4
58A418255AA")
1000 CLEAR&HFF00:SCREEN1,0:CLS:CFLASH:CSIZE:PRW&B1100:PALET2,7
1010 COLOR7:CGEN1:LINE(0,0)-(39,24),"A",BF
1020 FORI=0TO 9:FORJ=0TO15:A=I*16+J
1025 IF A>&HA AND A<&H10 OR A>&H1A AND A<&H20 OR A>&H2A AND A<&H
30 OR A=32 THEN GOTO1029
1027 LOCATEJ+14,I+10:PRINT#0CHR$(I*16+J)
1029 NEXT:NEXT:SCREEN0,0
1030 REPEAT:Z$=INKEY$:UNTIL Z$<>""
```

〈オオギ　1〉

```
10 ′ オ オ ギ
20 TEMPO 256:SOUND11,20:SOUND12,10:SOUND13,0:INIT
30 M$(0)="V1605C304BAB"
40 M$(1)="V1605C304BAE"
50 WIDTH40:SCREEN0,0:CLS4
60 FOR X=40 TO 280 STEP 80
70   FOR Y=70 TO 190 STEP 60
80     FOR I=7 TO 1 STEP -1
90       CIRCLE(X,Y),40,I,1,90-I*10,90+I*10
100        X1=40*COS(RAD(270-I*10))+X : Y1=40*SIN(RAD(270-I*10))+
Y
110        X2=40*COS(RAD(270+I*10))+X : Y2=40*SIN(RAD(270+I*10))+
Y
120        LINE(X,Y)-(X1,Y1),PSET,I
130        LINE(X,Y)-(X2,Y2),PSET,I
140          PAINT(X,Y-10),I,I
150        LINE(X,Y)-(X1,Y1),XOR ,I
160        LINE(X,Y)-(X2,Y2),XOR ,I
170      NEXT I
180    NEXT Y
190 NEXT X
200 FOR I=7 TO 1 STEP -1
210   PALETI,0 : PLAY M$(I MOD 2)
220 NEXT I
230 FOR I=1 TO 7
240   PALET I,I: PLAY M$(I MOD 2)
250 NEXT I
260 FOR I=1 TO 7
270   FOR K=7 TO 1 STEP -1
280     PALETK,0:PAUSE1
290   NEXT K
300   FOR J=1 TO 7
310     PALETJ,I:PAUSE1
320   NEXT J
330 NEXT I
340 GOTO200
```

〈オオギ 2〉

```
10 ' オ オ ギ
20 TEMPO 256:SOUND11,20:SOUND12,20:SOUND13,0:INIT
30 M$(0)="V1605C304BAB"
40 M$(1)="V1605C304BAE"
50 WIDTH40:SCREEN0,0:CLS4
60 FOR X=40 TO 280 STEP 80
70   FOR Y=70 TO 190 STEP 60
80     FOR I=7 TO 1 STEP -1
90       CIRCLE(X,Y),40,I,1,90-I*10,90+I*10
100        X1=40*COS(RAD(270-I*10))+X : Y1=40*SIN(RAD(270-I*10))+
Y
110        X2=40*COS(RAD(270+I*10))+X : Y2=40*SIN(RAD(270+I*10))+
Y
120        LINE(X,Y)-(X1,Y1),PSET,I
130        LINE(X,Y)-(X2,Y2),PSET,I
140          PAINT(X,Y-10),I,I
150        LINE(X,Y)-(X1,Y1),XOR ,I
160        LINE(X,Y)-(X2,Y2),XOR ,I
170      NEXT I
180    NEXT Y
190 NEXT X
200 FOR I=7 TO 1 STEP -1
210   PALETI,0 : PLAY M$(I MOD 2)
220 NEXT I
230 FOR I=1 TO 7
240   PALET I,I: PLAY M$(I MOD 2)
250 NEXT I
260 FOR I=1 TO 7
270   FOR K=7 TO 1 STEP -1
280     PALETK,0:PAUSE1
290   NEXT K
300   FOR J=1 TO 7
310     PALETJ,I:PAUSE1
320   NEXT J:PLAY "V1604A5AR7A5AR7:V1604B5B05B704B5B05B7"
330 NEXT I
340 PALET:GOTO200
```

エラーメッセージ便利表

エラーコード	エラーメッセージ	説　明
54	Already open	すでにOPENしてあるファイルを再びOPENしようとすると起こる。ファイルをOPENしたままプログラムを止めた場合、RUNし直すと一度すべてのファイルがCLOSEされ、GOTO文などでプログラムを再開すると起こることがある。
64	Bad allocation table	外部記憶装置には、記録されているファイル情報（FILESで表示されるもの）を記録しておくFATというものがあるが、これがこわれていると起こる。〝EMM：″や〝MEM：″をINITで初期化しないまま使おうとすると発生。
65	Bad file descripter	ディスクリプタが違う。
30	Bad file mode	X１で作成されるファイルには、BASICのテキスト、機械語のバイナリ、PRINT＃で作られるアスキー形式の３つのファイルがある。このエラーはPRINT＃で作ったファイルをLOADで読み込もうとすると起こる。ファイルの形式を調べたいときは、FILESを使うとよい。
52	Bad file number	OPENされていないファイルや、はじめから指定していないファイルを使おうとすると起こる。
25	Bad screen mode	OPTION SCREEN 2を指定せずに、グラフィックメモリを外部記憶としてINIT〝MEM；″などを実行すると起こる。また、80文字モードで、ASKを実行しようとしても起こる。
17	Can't continue	CONT文でプログラムの実行を再開できない。たとえば、RUNさせる前や、エラーが発生してコマンドレベルに戻ってしまった場合など。このような場合は、GOTO文によって先に進むことは可能。またFOR～NEXTなどをダイレクトモードで実行しているときに、BREAKした場合にもCONTは使えず、このエラーになる。
60	Device full	外部記憶装置の容量がいっぱいになったときに起こる。
51	Device in use	外部装置が使用中。この場合には、コマンドレベル、ステートメントレベルにならないので普通発生しない。
56	Device I/O ERROR	入出力装置において入出力エラーが生じた致命的エラー。〝MEM；″や〝EMM；″に対して、VERIFYを行なうと発生するが、これらのデバイスへSAVEした場合は、信頼性が高いので、VERIFYの必要はない。
73	Device Offline	入出力装置が接続されていないか、接続されていても電源が入っていない場合に起こる。
11	Division by zero	数学では、何かの値を０で割ると答えは無限大になるが、コンピュータでは扱えないので、このエラーで止まる。
10	Duplicate Definition	配列宣言を二度行なおうとしたときに起こる。プログラムで、実行を前の方に戻すときに、DIM文を二度通らないよう、戻り先の行を決めること。また、DEF FN文を二度実行しようとしても起こる。
50	FIELD overflow	FIELD文で、ランダムファイルのレコード長が256以上のときに起こる（ディスクBASIC）。

57	File already exists	NAMEでファイルネームを変更しようとしたとき、同じファイルネームがすでに登録されていると起こる（ディスクBASIC）。
53	File not found	ディスク、外部RAMと、グラフィックメモリを外部記憶として使っているとき、これらのデバイスにないファイルを使おうとすると起こる。
71	File not open	OPEN文で開かれていないファイルを使おうとすると起こる。
36	Format over	PRINT USINGで書式を指定してあるときに、データが長すぎて書式中に収まりきらないと起こる。
35	FOR without NEXT	FORループにNEXTがない。
12	Illegal direct	ダイレクトモードから実行できないステートメントを実行しようとすると起こる。X 1ではほとんど発生しない。
21	Illegal format	エラーメッセージの定義されていないエラーの場合起こる。しかし、めったに起こらず、起こってもUmprintable errorとなることがほとんど。
5	Illegal function call	LOCATE文やCONSOLE文などのように、オペランドのあるステートメントで、パラメータが規定外の数値になったときに起こる。
61	Input past end	end of fileのファイルを読もうとしたり、空のファイルを読もうとしたときに起こる。特に、PRINT＃1、A$;B$;C$;とした場合には、文字列は続けて書き込まれてしまうので、INPUT＃1、X$、Y$、Z$、として読み込もうとX$にA$、B$、C$が入ってしまい、このエラーになる。
23	Line buffur overflow	1行の入力文字数が多すぎる。
22	Missing operand	X 1のBASICでは、オペランドのパラメータを省略できるものとできないものがある。CREV、CONSOLEなどが省略できるもの、LOCATEなどができないものである。また、演算子の後に定数や変数がないときにもこのエラーになる。
1	NEXT without FOR	FORがないのにNEXTがある。
67	No password	パスワードと合わない。ファイルを作るときに、パスワードを設定することで、シークレットファイルを作ることができる。
19	No RESUME	エラー処理ルーチンからRESUMEでプログラムを再開しようとしても、できない場合に起こる。
4	Out of DATA	READ文でデータを読み込んでいくときに、DATAがたりないと発生。
7	Out of memory	プログラムが大きくなりすぎてメモリがたりないときや、配列が大きすぎるときなどに起こる。たとえば、次のような場合、 10　DIMA (16, 255) 20　FOR I＝1 TO 255 30　READ A 40　NEXT プログラム自体は49バイトしか使わないが、10行でDIM文が実行されると配列を格納するために、約21キロバイトのメモリが使用され、このままでDATA文を入力していけば、このエラーになる。このエラーが起こるとそれまでのすべての変数がクリアされてしまうので注意が必要。

31	Out of stack	このエラーは、今のところPUSH、POPというステートメントがないため、発生しない。
27	Out of tape	SAVE、LOADなど、カセットテープを使う命令を実行したときにテープがセットされていない。
6	Overflow	計算の結果が許容範囲を越えた。指数形式で表わされた場合、指数部が38を越えると起こる。
37	REPEAT without UNTIL	REPEATループにUNTILがない。
34	Reserved feature	テープ版BASICでディスクBASICのコマンドを使うと起こる。
20	RESUME without error	エラーが発生していないのにRESUME文があった場合に起こる。RETURN without GOSUBと同じように、プログラムの流れがエラー処理ルーチンに入り込まないよう、END文を入れる位置に注意。
3	RETURN without GOSUB	GOSUBで呼ばれていないのにRETURN文がある。よく起こるのはサブルーチンがプログラムの後の方にあって、メインルーチンを実行後にサブルーチンが実行されてしまう。このような場合には、メインルーチンの最後にEND文を入れて置かねばならない。
15	String too long	一つの文字列には最大255文字まで代入できるが、これを越えて代入しようとすると起こる。

9	Subscript out of range	配列変数の添字が規定外になった。添字の値が変数なら、その値を計算している部分を再チェックしよう。
2	Syntax error	文法のミス。ステートメントのつづりが違う場合がほとんど。記憶のあいまいなステートメントは短縮形を使うとよい。
29	Tape read ERROR	カセットテープからデータが正しく読めない。
16	Too complex	式が複雑すぎる。式の計算順序を決めるエリアも限りがあるため、カッコが異常に多い場合などに起きる。
13	Type mismatch	変数の型が合わない。次の例のような場合もある。LINE(0, 0)−(100, 100), A
18	Undefined function	DEF FNで定義していない関数を使った場合に起こる。DEF FN文は、プログラムの前の方に置くようにしよう。
8	Undefined label	GOTO文やGOSUB文で指定されている分岐先の行番号やラベルがない。
26	UNTIL without REPEAT	REPEATがないのにUNTILを実行しようとした。
33	WEND without WHILE	WHILEがないのにWENDがある。
32	WHILE without WEND	WENDがないのにWHILEがある。
72	Write protected	誤消去防止用のツメの折られているカセットテープに書き込みを行なおうとした。

上位4ビット→
下位4ビット↓
0123456789ABCDEF

SHARPパソコンテレビ

X1テクニカルマスター

定価はカバーに記載
されております

昭和58年10月5日　初版印刷
昭和58年10月8日　初版発行

著　者　ストラットフォードC.C.C.

発行者　孫　正義

発行所　株式会社日本ソフトバンク
出　版　部
千代田区四番町2-1(〒102)
☎03(261)4095

イラスト　ブーピープスタジオ 山田晴久

印　刷　凸版印刷株式会社

Printed in Japan　　ISBN4-930795-03